Approximate Multi-Agent Planning in Dynamic and Uncertain Environments

by

Joshua David Redding

B.S. Mechanical Engineering
Brigham Young University (2003)
M.S. Mechanical Engineering
Brigham Young University (2005)

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Aeronautics and Astronautics
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

December 2011

[February 2012]

Author ....
Department of Aeronautics and Astronautics
December 8, 2011

Certified by ....
Jonathan P. How
Richard C. Maclaurin Professor of Aeronautics and Astronautics
Thesis Supervisor

Certified by ....
Emilio Frazzoli
Associate Professor of Aeronautics and Astronautics

Certified by ....
Nicholas Roy
Associate Professor of Aeronautics and Astronautics

Accepted by ....
Eytan H. Modiano
Professor of Aeronautics and Astronautics
Chair, Committee on Graduate Students

Approximate Multi-Agent Planning in Dynamic and Uncertain Environments

by

Joshua David Redding

Submitted to the Department of Aeronautics and Astronautics
on January 26, 2012, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Aeronautics and Astronautics

Abstract

Teams of autonomous mobile robotic agents will play an important role in the future of robotics. Efficient coordination of these agents within large, cooperative teams is an important characteristic of any system utilizing multiple autonomous vehicles. Applications of such a cooperative technology stretch beyond multi-robot systems to include satellite formations, networked systems, traffic flow, and many others. The diversity of capabilities offered by a team, as opposed to an individual, has attracted the attention of both researchers and practitioners in part due to the associated challenges such as the combinatorial nature of joint action selection among inter-dependent agents. This thesis aims to address the issues of the issues of scalability and adaptability within teams of such inter-dependent agents while planning, coordinating, and learning in a decentralized environment. In doing so, the first focus is the integration of learning and adaptation algorithms into a multi-agent planning architecture to enable online adaptation of planner parameters. A second focus is the development of approximation algorithms to reduce the computational complexity of decentralized multi-agent planning methods. Such a reduction improves problem scalability and ultimately enables much larger robot teams. Finally, we are interested in implementing these algorithms in meaningful, real-world scenarios. As robots and unmanned systems continue to advance technologically, enabling a self-awareness as to their physical state of health will become critical. In this context, the architecture and algorithms developed in this thesis are implemented in both hardware and software flight experiments under a class of cooperative multi-agent systems we call persistent health management scenarios.

Thesis Supervisor: Jonathan P. How
Title: Richard C. Maclaurin Professor of Aeronautics and Astronautics

Acknowledgments

Although none of my daughter's kindergarten classmates believed her when she told them her daddy went to *school* every morning instead of *work*, and although my son has asked "Who's that guy standing next to Mommy?" in several family pictures, and although it has meant that my sweet wife has been a single mother for far too many diaper changes - I hope that my family will someday find that we made the right choice in forgoing any resemblance of a livable income for a few too many years and sending me back to school. Yes, the Aerospace Controls Lab has literally been my second home for four incredible years. Many people have contributed not only to the completion of this thesis, but also to the enjoyment and productivity of these four years. Foremost among these is my sweetheart, Erica. Her endless patience (well, *almost* endless!), heart-warming smile and deep support made it both possible and enjoyable to wade through the challenges of earning a PhD. Between her and our kids, my little family is an immeasurable source of real happiness and of real motivation.

Second, I'd like to acknowledge my advisor, Jonathan How, for his vision, his talent for research, and yes, for his patience also. Somehow he always seemed to know just the right questions to ask to prod my research in the right direction. Equally important was his remarkable ability to tactlessly tear my research apart without being offensive, then follow up with positive and insightful suggestions regarding a path forward. However, even with the many ups and downs, working in Jon's lab was really fun for me (often to the point of distraction) and I feel lucky to have had him as an advisor.

Third, I'd like to recognize Boeing Research and Technology for their generosity in funding my research from start to finish. Specifically, I'd like to thank Matt Vavrina and John Vian for the time they sacrificed to listen to, plan for, discuss and support my research on both theoretical and practical levels. It has been a pleasure working with both of them.

Finally, down in the trenches, I'd like to formally thank my amazing lab-mates and publicly acknowledge that this thesis would not have been possible without their help. Specifically, I owe a special debt of gratitude to: Brett Bethke, who is arguably the best teacher I have ever had (well, academically anyway) and whose smile is unceasing; Buddy Michini, who posesses a rare blend of confidence, tenacity and incredible aptitude and who is always willing to lend a hand (or mind); Tuna Toksöz, whose work ethic is truly inspirational and whose coding skills are second-to-none, he is a lifelong friend; Kemal Üre, whose sharp math skills and friendly personality will take him far; Alborz Geramifard, who understands all things Markovian better than anyone I know and happily shares that knowledge to the benefit of others; Sameera Ponda, whose ability to work and think under pressure is unsurpassed; Dan Levine, whose sense of humor I will really miss, along with his knowledge of mathematical notation and English grammar; Luke Johnson, who has the amazing ability to fundamentally understand just about any tricky concept you throw at him, and who can easily make me smile; Brandon Luders, whose help I could always count on, from theory to implementation; Frank Fan, whose knowledge of Matlab, Fortran, LaTeX, control theory, motorcycles and fish runs deep and with whom I've shared many a laugh; Aditya Undurti, who basically taught me how to code and who has a knack for explaining probability theory and Kalman filtering; Andy Whitten, whose working knowledge of all things R/C and contagious optimism made me a better engineer; Vishnu Desaraju, who is probably remotely connected to a handful of computers right now - all through a virtual machine running on his phone; Andrew Kopeikin, who has a real talent for commnication, both in person and on paper; and Mark Cutler, who has hit the ground running with a keen attention to detail, a strong work ethic and polished Matlab skills. To these and others, thank you for the lessons, the laughs and the memories.

Contents

List of Figures

Chapter 1

Introduction

Life is full of decisions. Many are trivial and inconsequential, while others complex with serious and far-reaching implications. As humans, we often match the time spent making a decision with our perception of its long-term implications. For example, deciding what to have for breakfast typically takes only a few moments, while deciding which college to attend, or which job to take, can take months. Robots are no exception. Having become critical tools to warfighters, first-responders, homeland security agents and quality control inspectors, unmanned robotic systems are faced with a variety of decisions and must also balance evaluation with action while accomplishing their objectives.

The use of robotic and unmanned systems has been rising sharply for more than a decade and, so far, there is no plateau on the horizon. In fact, in 2009 the U.S. Air Force started training more pilots to operate unmanned systems than to physically fly fighters and bombers [110]. In addition, the U.S. Congress has mandated that by the year 2015, one-third of all ground combat vehicles will be unmanned [110]. This increase in the use and development of robotic and unmanned systems is largely due to their ability to amplify operator capability while simultaneously reducing costs and/or risks. The potential for this ability was seen decades ago with the development of pilot-less "aerial torpedoes" of WWI [41], and of "Shakey" in the late sixties, which is arguably the first major mobile ground robot development effort recorded [37].

Despite the incredible advances in the technology behind these frontrunners, the concept of multiple unmanned vehicles cooperating seamlessly with both manned and unmanned platforms simultaneously remains a formidable barrier. One of the primary challenges in overcoming this barrier is trust. Psychological reasons aside, our inability to trust an unmanned system is evidenced by the fact that the current operator/robot ratio sits near five to one. To invert this ratio, robot predictability must be thoroughly tested and proven. This presents major challenges not only in human-machine interfaces, but also in the verification and validation of the algorithms running onboard the robotic systems. With regard to the latter, a fundamental question that remains unanswered is "How do you test a fully autonomous system?" Similarly, "How can one predict a robot's response to a situation for which it was not designed, or pre-programmed?" To date, there is no accepted way of subjecting an autonomous system to every conceivable situation it might encounter in the real world [110]. Even lowering our expectations a notch, what statistically meaningful tests are there that will lead to acceptable confidence in the behavior of these autonomous systems? Only as the predictability of such systems is proven beyond current levels will our trust grow enough to invert the operator/robot ratio.

A second challenge is interoperability. As a case in point, the Army's unmanned system A, cannot seamlessly interact with the Navy's robotic system B, particularly when the systems are not built by the same contractor [110]. One reason for this may be that unmanned ground and maritime systems typically use a messaging standard called the Joint Architecture for Unmanned Systems (JAUS) [52], while unmanned aerial systems use a NATO-mandated STANAG-4586 [54] protocol. However, recent efforts in this area have produced the promising JAUS toolset [55], an open, standards-based messaging suite currently in beta testing[110]. Additionally, the Joint Unmanned Aircraft Systems Center of Excellence [56] is also working toward developing communication standards to encourage interoperability between systems from different vendors.

Another challenge for unmanned systems is computation. This is a broad challenge, with facets including processing raw sensor data to planning sequences of future actions. To put this in context, during 2009, U.S. unmanned aerial vehicles (UAVs) sent 24 years' worth of video footage back to operators, a number that is expected to escalate to nearly 720 years' worth in 2011 [110]. Needless to say, this number doesn't exactly make you want to reach for the popcorn. Similarly, making decisions to coordinate actions and plans within a team of robotic agents is combinatorial in nature and thus requires an extreme amount of computation to search through all possible combinations - quickly becoming as complex as the 720 years of video, even for teams as small as two agents. Thus, *scalability* is another branch of this challenge as planning for teams of robotic agents typically grows exponentially in the number of agents.

This thesis aims, in part, to address this issue of scalability. Specifically, one focus is on the development of approximation algorithms to reduce the computational complexity of decentralized multi-agent planning methods, thereby improving their scalability and ultimately enabling larger robot teams. Another focus of this thesis is the integration of learning algorithms into a multi-agent planning architecture that will enable online adaptation of planner parameters. Finally, we are interested in implementing these algorithms in meaningful, real-world scenarios. As robots and unmanned systems continue to advance technologically, enabling a self-awareness as to their phyisical state of health will become critical. In this context, we focus on a specific class of cooperative multi-agent systems we call persistent health management (PHM) scenarios.

1.1 Problem Description and Solution Approach

In the context of multiple coordinating agents, many mission scenarios of interest are inherently long-duration and require a high level of agent autonomy due to the expense and logistical complexity of direct human control over individual agents. Hence, autonomous mission planning and control for multi-agent systems is an active area of research [30, 42, 48, 72, 74, 94, 95].

Long-duration missions are practical scenarios that can show well the benefits of agent cooperation. However, such *persistent* missions can accelerate mechanical wear and tear on an agent's hardware platform, increasing the likelihood of related failures. Additionally, unpredictable failures such as the loss of a critical sensor or perhaps damage sustained during the mission may lead to sub-optimal mission performance. For these reasons, it is important that the planning system accounts for the possibility of such failures when devising a mission plan. In general, planning problems that coordinate the actions of multiple agents, where each of which is subject to failures are referred to as multi-agent *persistent health management* problems [106, 107]. There are two typical approaches for dealing with PHM problems [22]. The first is to construct a plan based on a deterministic model of nominal agent performance. Though computationally inexpensive, this approach ignores the possibility of random, unplanned events and simply computes a new plan when measured performance deviates further than expected. Since the system does not model failures, it cannot anticipate them, but rather responds to them once detected, this approach is referred to as *reactive*.

In contrast, a second, *proactive* approach to planning constructs plans based on internal, stochastic models which aim to capture the inherent possibility of certain failures. Using stochastic models increases the complexity of computing the plan, as it then becomes necessary to optimize *expected* performance, where the expectation is taken over all possible combinations of scenarios that might occur. However, since proactive planners dictate actions that aim to mitigate the consequences of possible future failures, the resulting mission performance can be much better than that achieved by a reactive planner. Naturally, this potential performance increase depends on the validity of the underlying stochastic models. When there exists uncertainty around these models or their parameters, performance can suffer.

While cooperative, proactive planners for PHM missions can be formulated in a number of ways (e.g. LP, MILP, heuristic [4, 61–63]), this thesis adopts a dynamic programming approach. While further background is provided in Chapter 2, dynamic programming presents a natural framework for formulating stochastic, sequential decision making problems, such as the cooperative planning problem.

This thesis has three main areas of focus. The first is to develop an architecture that enables learning within multi-agent PHM systems. Learning algorithms provide a mechanism for the planner to reduce uncertainties in the underlying model by tuning the associated parameters online using accumulated sensor and performance data. Second, this thesis introduces several approximation algorithms aimed to reduce the complexity (and thus, increase the scalability) of multi-agent planning algorithms. The goal of these approximations is to sufficiently reduce the computation of the multi-agent planning problem such that an online solution becomes possible which can fully utilize the benefits of the learning mechanisms within the architecture discussed previously. The third focus of this thesis is to demonstrate and validate the architecture and approximate planning algorithms through both simulation and flight tests involving multiple, coordinating agents in a realistic PHM scenario.

1.2 Summary of Contributions

As mentioned, one of the three focus areas of this thesis is the integration of multi-agent planning algorithms with online adaptation, or learning, algorithms within a common architecture. The motivation for such an architecture is that, in many applications, the basic form of models internal to the planner are known, while their associated parameters are not. As a result, the uncertainty around these parameters may lead to significant suboptimal performance in the system. This is particularly evident when the resulting control policy from an MDP-based multi-agent planner (one that was modeled using uncertain parameters) is actually implemented on the multi-agent system. In addition, these parameters may well be time-varying, which introduces further sub-optimalities, especially if the planning problem is too large to be solved online. To address these issues, an architecture for continuous online planning, learning and replanning is developed that combines a generic multi-agent planner with parameter learning and performance evaluation algorithms. With regard to the development of such an environment, this thesis contributes the following:

- A template architecture is developed to enable the integration of multi-agent cooperative control techniques with online learning algorithms and performance evaluation metrics. We call the result the *intelligent Cooperative Control Architecture*, or iCCA. As a modular template, iCCA permits the use of a variety of multi-agent planning algorithms, learning methods and evaluation functions.

- The iCCA template is instantiated under multiple combinations of planner, learner and evaluation components and is implemented in both simulation and flight-test environments. Planner types include MDPs and a market-based iterative auction algorithm called the consensus-based bundle algorithm (CBBA). Under the learning component, techniques such as direct adaptive control, maximum likelihood estimation and reinforcement learning are included.

A second focus of this thesis is the development of approximation algorithms for the decentralized control of multiple agents in a cooperative environment. The motivation for such approximations is primarily to avoid the exponential explosion of required calculcations when considering the combinatorial coupling between participating agents under an MDP-based planner. This thesis addresses this challenge by allowing each agent to represent its teammates with an approximate model while maintaining a full-fidelity model of itself. Thus, when viewed collectively, the team contains a full model for each agent while the problem size for each agent remains small enough that a solution is computable within the timescale of the planning horizon (typically seconds, or minutes). We call our approach a *decentralized multi-agent Markov decision process (Dec-MMDP)* to emphasize the connection with the well-studied Multi-agent Markov decision process (MMDP) while first identifying the decentralized nature of the problem formulation. With regard to the development of these approximation algorithms, this thesis makes the following contributions:

- The Dec-MMDP algorithm is developed which introduces a mechanism for reducing problem size by approximately representing individual teammates using a *feature-based* state aggregation technique on the full agent model. A *feature* is an abstraction of a set of states and actions. For example, a binary feature "IsHealthy" is a simple abstraction of a potentially much larger, much more complex set of states and actions that include state-dependent fuel and actuator models. We analytically show that this algorithm results in a computational complexity that now scales as $|D|^{n-1}$ in the number of agents n rather than $|C|^n$, where $D << C$. Also, we empirically show that the resulting performance is within 10% of that achieved by its centralized counterpart.

- An extension to the basic Dec-MMDP formulation is developed where each agent now models the aggregate set of its teammates rather than each teammate individually. We call this extension the group-aggregate Dec-MMDP (GA-Dec-MMDP). We analytically show that this extension results in a computational complexity that can be made to scale *linearly* in the number of agents n, rather than exponentially. Also, we empirically show that the resulting performance is within 20% of that achieved by its centralized counterpart.

- A method is developed for generating the feature vector, ϕ, used in the approximation algorithms listed above. We show how the components of the cost/reward function are translated into features, either binary or $(n-1)$-ary. Also, we analytically show how the resulting problem complexity is affected through construction of these features.

A third focus area deals with the autonomous planning in multi-agent robotic systems. In particular, we investigate the persistent surveillance mission (PSM), in which multiple unmanned aerial vehicles (UAVs) and/or unmanned ground vehicles (UGVs) must continuously search a designated region over indefinite periods of time. This problem directly relates to a number of applications, including search and rescue, natural disaster relief operations, urban traffc monitoring, etc. These types of missions are challenging largely due to their extended duration, which increases the likelihood that one or more agents will experience health-related failures over the course of the mission. The planning system must anticipate and plan for these failures while maximizing mission performance. In order to investigate these issues, this thesis:

- Formulates the persistent surveillance mission as an MMDP, which fully captures the requirement of scheduling agents to periodically move back and forth between the surveillance location and the base location for refueling and maintenance. In addition, we incorporate a number of randomly-occurring failure scenarios (such as sensor failures, actuator degradations and unexpected fuel usage) and constraints (such as the requirement to maintain a communication link between the base and agents in the surveillance area) into the problem formulation. We show that the optimal policy for the persistent surveillance problem formulation not only properly manages asset scheduling, but also anticipates the adverse effects of failures on the mission and takes proactive actions to mitigate their impact on mission performance.

- Formulates the persistent surveillance mission as a Dec-MMDP and as a GA-Dec-MMDP. We empirically show that the resulting policies properly manage asset scheduling while anticipating the adverse effects of failures and attempt to take proper proactive actions so as to mitigate their impact on mission performance. We further show that implementing the resulting policies in the persistent surveillance scenario results in mission performances within 10% and 20% of the optimal solution, respectively.

- Highlights the development of an automated battery changing/charging station for hover-capable unmanned aerial vehicles. This technology is required for a truly autonomous persistent capability.

1.3 Thesis Outline

The remainder of this thesis is outlined as follows: Chapter 2 presents background material on multi-agent systems, dynamic programming, Markov decision processes (MDPs) and reinforcement learning. Next, Chapter 3 discusses the details of the intelligent cooperative control architecture (iCCA) and motivates the need for reduced-complixity planning methods that can be solved online. Chapter 4 presents several approximation algorithms designed to exploit teammate structure in a multi-agent setting in order to reduce the compuational complexity of solution generation to within the limits of online calculation. Specifically, the decentralized and group-aggregate decentralized multi-agent Markov decision processes are formulated. A particular PHM scenario known as the persistent surveillance mission (PSM) is outlined in Chapter 5, and both simulation and flight-test results are given in Chapter 6 as both approximation algorithms are implemented under the iCCA framework. Finally, conclusions are offered and future work is suggested in Chapter 7.

Chapter 2

Background

The purpose of this chapter is to provide the foundational underpinnings of the remainder of this thesis. The sections that follow provide the conceptual and algorithmic building blocks for the topics of cooperative multi-agent systems, multi-agent planning, and reinforcement learning. Where appropriate, relevant work from the existing body of literature is cited and an effort is made to tie it to the scope, purpose and contributions of this thesis.

2.1 Cooperative Multi-Agent Systems

To begin, we address the question "What is an agent"? It turns out there are a handful of definitions centering around an "autonomous entity". For example, in the aritificial intelligence community, an agent can be either physical or virtual, but must be able to perceive at least some of its environment, communicate with other agents, and act in order to achieve its goals [34, 91]. In computer science, agents typically assume a more virtual, or software-based, role while in cooperative control an agent commonly refers to a physical robot with tangible sensors and actuators. Russell and Norvig [91] provide a directory of agent types, two of which are most relevant to this thesis: utility-based and learning agents.

Utility-based agents are distinguished by their ability to calculate the "utility" or "value" of a particular state. This allows for comparisons among reachable states that aids the decision of which action to take. A rational utility-based agent chooses the action that is expected to result in the most value, where this expectation may utilize models of the environment and the agent's dynamics [91].

Learning agents have the advantage of becoming more competent over time while acting in unknown, or partially known, environments. A learning agent is most easily thought of as comprised of a performance and a learning element. The learning element distinguishes this type of agent above the others by using observations regarding its current performance (e.g. in the form of temporal difference errors) to make improvements on future performance by modifiying the performance element itself [91].

The remainder of this thesis considers an agent to be both utility-based and learning, as in the case of Bayesian reinforcement learning [102]. However, we also consider an agent to be a physical, autonomous entity capable of perceiving, communicating and acting in order to achieve its goals. Given this notion, a cooperative multi-agent system (CMAS) is an environment with multiple communicating agents whose actions change the perceivable state of the environment [34]. Sources of uncertainty within a CMAS include sensor noise, agent action outcome, teammate action choice and teammate action outcome.

2.2 Multi-Agent Planning

The overall challenge of planning in a CMAS is selecting coordinating actions between all agents in the presence of the aforementioned uncertainties.There are, in general, two basic branches of cooperative planning for addressing this challenge: centralized and decentralized planning. In a centralized approach, a single agent collects available information and generates/dictates the actions, or plans, of all other participating agents. Essentially, centralized approaches treat the whole team as a single agent as all information is known. In general, centralized algorithms tend to be conceptually simpler, easier to optimize, but less robust to failures [8].

In a decentralized approach, each agent decides for itself what actions it will take or what plans it will implement, based on information perceived and/or communicated to it. While this approach has the benefit of excluding single-point-of-failure weaknesses, one particular challenge that does not arise in centralized architectures is that each agent has no guarantee on what actions or plans its teammates might implement. Now, in certain cases, e.g. identical agents, there are methods of dealing with this issue. One such approach is known as *implicit coordination* [39] and involves each agent solving an identical, centralized problem with a mutually agreed upon input. In these cases, the same input is run through the same algorithm and results in the same predictable output. The remainder of this thesis is restricted to the common case where the agents in a CMAS run identical algorithms, but do not necessarily experience identical inputs. This is known as *explicit coordination* [39] and means that communication (and possibly iteration) is required to acheive agreement, or coordination, among the agents' plans. The subsections that follow are intended to provide sufficient background information regarding several algorithms used later in this thesis for formulating the decentralized cooperative multi-agent planning problem under explicit coordination.

2.2.1 Consensus-Based Bundle Algorithm

The consensus-based bundle algorithm (CBBA) is an approximate, decentralized, multi-agent market-based task-assignment algorithm that alternates between two phases: Bundle construction and Deconfliction. In the first phase, each vehicle generates a single, ordered "bundle" of tasks by sequentially adding the task that yields the largest expected marginal score. The second phase resolves inconsistent or conflicting assignments through local communication between neighboring agents [26]. The algorithm iterates between these two phases until agreement is reached on the assignments.

Specifically, given a list of N_t tasks and N_a agents, the goal of the task allocation algorithm is to find a conflict-free matching of tasks to agents that maximizes some global reward. An assignment is said to be free of conflicts if each task is assigned to no more than one agent. The global objective function is assumed to be a sum of local reward values, while each local reward is determined as a function of the tasks assigned to that particular agent. The task assignment problem described above can be written as the following integer (possibly nonlinear) program:

$$\begin{aligned}
\max \quad & \sum_{i=1}^{N_a}\left(\sum_{j=1}^{N_t} c_{ij}(\tau_{ij}(\mathbf{p}_i(\mathbf{x}_i)))x_{ij}\right) \\
\text{subject to:} \quad & \sum_{j=1}^{N_t} x_{ij} \leq L_t, \ \forall i \in \mathcal{I} \\
& \sum_{i=1}^{N_a} x_{ij} \leq 1, \ \forall j \in \mathcal{J} \\
& x_{ij} \in \{0,1\}, \ \forall (i,j) \in \mathcal{I} \times \mathcal{J}
\end{aligned} \tag{2.1}$$

where the binary decision variable x_{ij} is 1 if agent i is assigned to task j, and $\mathbf{x}_i \in \{0,1\}^{N_t}$ is a vector whose j-th element is x_{ij}. The index sets are defined as $\mathcal{I} \triangleq \{1, \ldots, N_a\}$ and $\mathcal{J} \triangleq \{1, \ldots, N_t\}$. The vector $\mathbf{p}_i \in (\mathcal{J} \cup \{\emptyset\})^{L_t}$ represents an ordered sequence of tasks for agent i; its k-th element is $j \in \mathcal{J}$ if agent i conducts j at the k-th point along the path, and becomes $\emptyset$ (denoting an empty task) at the k-th point if agent i conducts less than k tasks. L_t is a limit on the maximum amount of tasks that can be assigned to an agent. The summation term in brackets in the objective function represents the local reward for agent i. Key assumptions underlying the above problem formulation are:

- The score c_{ij} that agent i obtains by performing task j is defined as a function of the arrival time τ_{ij} at which the agent reaches the task (or possibly the expected arrival time in a probabilistic setting).

- The arrival time τ_{ij} is *uniquely* defined as a function of the path $\mathbf{p}_i$ that agent i takes.

- The path $\mathbf{p}_i$ is *uniquely* defined by the assignment vector of agent i, $\mathbf{x}_i$.

Many interesting design objectives for multi-agent decision making problems feature scoring functions that satisfy the above set of assumptions. The time-discounted value of targets [2, 11] is one such example, in which the sooner an agent arrives at the target, the higher the reward it obtains. However, CBBA allows for various design objectives, agent models, and constraints through the designer's choice of an appropriate scoring functions. If the resulting scoring scheme satisfies a certain property called *diminishing marginal gain* (DMG), a provably good feasible solution is guaranteed. Furthermore, the score function allows for time-discounted rewards, tasks with finite time windows of validity, heterogeneity in agent capabilities and vehicle-specific fuel costs, all while preserving the robust convergence properties [82].

2.2.2 Dynamic Programming

Dynamic pogramming is a mathematical framework for posing sequential decision problems, including multi-agent extensions. The basic framework incorporates two main elements: A system model and a cost function [12]. The system model describes the dynamics of the system, possibly including stochastic state transitions. The cost function depends on the system state and potentially also the decision made while in that state. The solution of a dynamic program is a *policy*, which essentially constitutes a state-dependent rule for choosing which action to take from any particular state so as to minimize the expected, discounted sum of the costs incurred as specified by the cost function. When the state space associated with a particular dynamic programming problem is discrete, it becomes a Markov decision process (MDP).

2.2.3 Markov Decision Processes

A Markov decision process (MDP) is a versatile mathematical framework for formulating stochastic, sequential decision problems. Common among the many flavors of MDPs are the concepts of state, action, transition and utility (also known as value). A state, in the MDP context, defines a current snapshot of the system and simulataneously summarizes its relevant history. The state space of an MDP is the space spanned by the union of all individual states. Action and transition are coupled in the sense that given an initial state s, an action a causes a transition of the system to a new state s'. The Markov assumption dictates that the probability of reaching a single state s' is a function of the previous state s and not of any other state. This essentially means that all the information needed for making intelligent decisions should be wrapped into the notion of the state. Now, from any given state of the system, the decision problem becomes "Which action do I take?". It is in addressing this question that the fundamental concept of utility, or value, arises. Utility is conceptually simple - it is a metric that defines the relative value of the system being in one state over the others in the state space. Mathematically, the utility of a given state, $J(s)$, is connected to the current policy, π, and cost function $g(s)$ as

$$\min_{\pi\in\Pi} J_\pi(s_0) = \min_{\pi\in\Pi} \mathbb{E}\left[\sum_{k=0}^{\infty} \alpha^k g(s_k, \pi(s_k))\right]. \tag{2.2}$$

A state's utility, $J(s)$, is also known as its "cost-to-go" or, the expected cost that the agent will receive when starting from that state and thereafter following some fixed policy and is calculated based on reward function and transition dynamics. The solution of an MDP is called a *policy* π, which specifies an action for every state the agent might reach. The optimal policy, π^*, specifies the best action to take at each state, i.e. the one that will lead to the highest *expected* utility over the planning horizon. An infinite-horizon, discounted MDP is specified by the following tuple: $\langle\mathcal{S}$, $\mathcal{A}$, P, g, $\alpha\rangle$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $P(s'|s,a)$ gives the probability of transitioning into state $s' \in \mathcal{S}$ given starting from state $s \in \mathcal{S}$ and taking action $a \in \mathcal{A}$, and $g(s,a)$ gives the cost of taking action a from state s. We assume that the model, P, is known. Future costs are discounted by a factor $0 < \alpha < 1$. The outcome of the MDP, e.g. solution, is a policy, denoted by $\pi : \mathcal{S} \to \mathcal{A}$, and is a mapping of states to actions. Given the MDP specification, the policy is found by minimizing the *cost-to-go* function J_π over the set of admissible policies Π, as shown in Equation (2.2) above.

The cost-to-go for a fixed policy π satisfies the Bellman equation [16]

$$J_\pi(s) = g(s,\pi(s)) + \alpha \sum_{s'\in\mathcal{S}} P(s'|s,\pi(s)) J_\pi(s') \quad \forall s \in \mathcal{S}, \tag{2.3}$$

which can also be expressed compactly as $J_\pi = T_\pi J_\pi$, where T_π is the (fixed-policy) dynamic programming operator [12, 13]. Further details regarding MDPs, their complexity and solution approaches can be found in [79, 91, 102].

2.2.4 Multi-agent Markov Decision Processes

A multi-agent Markov decision process (MMDP) is essentially a large MDP that simultaneously calculates decisions for more than one decision-maker. An MMDP generalizes the above-formulated MDP to multi-agent, cooperative scenarios. An infinite-horizon, discounted MMDP is specified by the following tuple: $\langle n, \mathcal{S}, \mathcal{A}, P, g, \alpha \rangle$, where n is the number of agents, $\mathcal{S}$ is now the *joint* state space, $\mathcal{A}$ is the *joint* action space, $P(s'|s,a)$ gives the probability of transitioning into *joint* state $s' \in \mathcal{S}$ from *joint* state $s \in \mathcal{S}$ having taken *joint* action $a \in \mathcal{A}$, and $g(s,a)$ gives the cost of taking action a in state s. In the general formulation, the joint state and action spaces are each fully-coupled and cannot be factored by agent. However, in this thesis we assume the agents to be transition-independent. That is, we assume the transition dynamics of one agent are completely de-coupled from, and therefore un-influenced by, those of any other agent. This assumption enables the joint state space to be factored as $\mathcal{S} = \prod \mathcal{S}_i \; \forall i \in \{1 \dots n\}$. Similarly, the joint action space becomes $\mathcal{A} = \prod \mathcal{A}_i \; \forall i \in \{1 \dots n\}$, where $\mathcal{S}_i$ and $\mathcal{A}_i$ are the local state and action spaces for agent i. It is important to note that if the cost/reward g is factorable, the agents are considered reward-independent and if this is in conjunction with transition-independence, then the MMDP becomes n separable MDPs. In this thesis, the agents' rewards are considered to be coupled.

2.2.5 Decentralized Markov Decision Processes

In an MMDP, each agent has *individual full observability* of the joint state. This means that each agent can itself observe the full, joint state. Hence, MMDPs do not model observations and have been shown to be NP-Complete [70]. Relaxing individual full observability just a notch, we arrive at *joint full observability*, which means that only the combined observations of all agents can represent the full joint state. Under this new assumption, we arrive at a decentralized Markov decision process (Dec-MDP), which is a generalization of the MMDP in the sense that local full observability is relaxed to joint full observability. Specifically, a Dec-MDP is specified by the following tuple: $< n, \mathcal{S}, \mathcal{A}, P, g, \alpha, \mathcal{Z}, O >$, where n is the number of agents, $\mathcal{S}$ is again the joint state space, $\mathcal{A}$ is again the joint action space, $P(s'|s,a)$ again gives the probability of transitioning to joint state s' when starting from joint state s and taking joint action a, and $g(s,a)$ again gives the cost of taking action a from state s. New to this formulation however, are the set $\mathcal{Z}$ and the model O. The set $\mathcal{Z}$ contains all joint observations and the model $O(z|s,a)$ gives the probability of receiving joint observation $z \in \mathcal{Z}$ given joint state s and joint action a. Under the assumption of joint full observability, for each joint state-action pair there exists a joint observation $z = \{z_1 \dots z_n\}$ such that $O(z|s,a) = 1$. The solution of a Dec-MDP is a policy that maps sequences of observations to a joint action.

It is important to note that in moving from a centralized to a decentralized formulation, the shift from *individual* to *joint* full observability causes the resulting problem complexity to jump from NP-Complete to NEXP-Complete [43], which includes the class of super-exponential problems. The primary reason for such a dramatic increase in computational complexity is the fact that now, agents must maintain, and reason over, a history of observations in order to infer a set of possible joint states that they might be in. For this reason, decentralized decision problems are considerably more difficult than their centralized counterparts - as all information cannot necessarily be shared at each time step. For further details regarding the complexity of Dec-MDPs, the interested reader is referred to Refs [15] and [43], where such details can be found.

Despite the added complexity, decentralized planning and control problems are ubiquitous. Examples include space exploration rovers, load balancing in decentralized queues, coordinated helicopters and sensor network management [95, 115]. The general problem in question here turns out to be a DEC-POMDP, where joint full observability is further relaxed to partial observability - that is, each agent may have different partial information about the state of the world. In either case however, the agents must cooperate to optimize some joint cost/reward function. If each agent were to have its own cost/reward function, the result is a partially observable stochastic game (POSG) [47].

Communication in a multi-agent setting can help reduce the general complexity from NEXP-Complete. When messages from teammates contain information about the state of the system, or about their intentions, the complexity can be reduced again to NP-Complete [95, 99]. Essentially, this amounts to using inter-agent communication to replace individual full observability. In this thesis, we restrict our scope to systems with individual full observability of the state, be it through observation or communication.

Chapter 3

An Intelligent Cooperative Control Architecture

Most applications of heterogeneous teams of autonomous robots require that participating agents remain capable of performing their advertised range of tasks in the face of noise, unmodeled dynamics and uncertainties. Many cooperative control algorithms have been designed to address these and other, related issues such as humans-in-the-loop, imperfect situational awareness, sparse communication networks, and complex environments [58, 68, 93].

While many of these approaches have been successfully demonstrated in a variety of simulations and some focused experiments, there remains room to improve overall performance in real-world applications. For example, cooperative control algorithms are often based on simple, abstract models of the underlying system. This may aid computational tractability and enable quick analysis, but at the cost of ignoring real-world complexities such as intelligently evasive targets, adversarial actions, possibly incomplete data and delayed or lossy communications.

Additionally, although the negative influence of modeling errors are relatively well understood, simple and robust extensions of cooperative control algorithms to account for such errors are frequently overly conservative and generally do not utilize observations or past experiences to refine poorly known models [1, 19]. Despite these issues however, cooperative control algorithms provide a baseline capability for achieving challenging multi-agent mission objectives. In this context, the following research question arises: *How can current cooperative control algorithms be extended to result in more adaptable planning approaches?*

To address this question while improving long-term performance in real-world applications, we propose an extension to the cooperative control algorithms that enables a tighter integration with learning techniques. Many learning algorithms are well suited for on-line adaptation in that they explicitly use available data to refine existing models, leading to policies that fully exploit new knowledge as it is acquired [20, 86]. Such learning algorithms, combined with a cooperative planner, would be better able to generate plans that are not overly conservative. However, learning algorithms are also prone to limitations, including the following:

- They may require significant amounts of data to converge to a useful solution
- Insufficient coverage of the training data can lead to "over-fitting" and/or poor generalization
- There are no guarantees on the robustness of the closed *learner-in-the-loop* system (robustness in learning algorithms typically refers to the learning process itself)
- Exploration is often explicit (e.g., by assigning optimistic values to unknown areas) which, in the context of cooperative control, can lead to catastrophic mistakes
- Scenarios where agents do not share complete knowledge of the world may cause the learning algorithm to converge to local minima or to fail to converge at all

In this thesis, we combine learning with an underlying cooperative control algorithm in a general, synergistic, solution paradigm called the *intelligent Cooperative Control Architecture* (*iCCA*), where some of these limitations can be addressed. Firstly, the cooperative planner can generate information-rich feedback by exploiting the large number of agents available for learning, addressing problems raised by insufficient data. Second, learning algorithms are more effective when given some prior

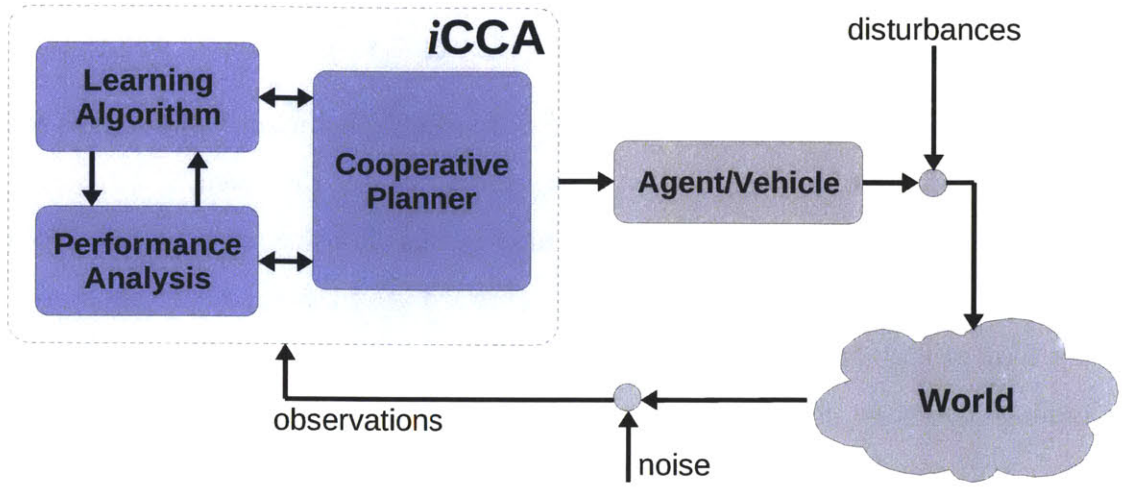


Figure 3-1: An **i**ntelligent **C**ooperative **C**ontrol **A**rchitecture designed to mitigate the effects of modeling errors and uncertainties by integrating cooperative control algorithms with learning techniques and a feedback measure of system performance.

knowledge to guide the search and steer exploration away from catastrophic decisions. A cooperative planner can offer this capability, ensuring that mission objectives are achieved even as learning proceeds. In return, the learning algorithm enhances the performance of the planner by offering adaptability to time-varying parameters. We later show that this combination of cooperative control and learning results in better performance and more successful executions of real-world missions.

Figure 3-1 shows the *intelligent Cooperative Control Architecture* (*iCCA*), that was designed to provide customizable modules for implementing strategies against modeling errors and uncertainties by integrating cooperative control algorithms with learning techniques and a feedback measure of system performance. The remainder of this chapter describes each of the *iCCA* modules followed by a sampling of example *iCCA* applications. Specifically, Section 3.1 discusses the cooperative control algorithm requirements, Section 3.2 describes the observations and performance metric(s) and Section 3.3 outlines the requirements and assumptions associated with the learning element of *iCCA*. Following these descriptions, Section 3.4 provides a few examples of the application of *iCCA*.

3.1 Cooperative Planner

The term *planning* carries several meanings. To the AI community, planning is searching for a sequence of actions that, when executed, enable the agent to achieve a particular goal. To a decision-theorist, planning means choosing actions that maximize expected reward (utility, payoff), where the reward is a function of current state, goal state and probability of action outcomes. From a control-theoretic point of view, planning involves calculating a sequence of control inputs to track a reference trajectory with minimal error. Underlying these points of view is the common notion of optimized *action selection.* In the context of multi-agent planning, it is common to interchangeably consider optimized *task* selection, as tasks are readily associated with actions. For planning purposes, the task assignment optimization must be able to quantify the overall cost of a task. For example, given a heterogeneous team of cooperating agents with combined state x, roles α and control input u, task cost can be expressed over horizon T as

$$J_{\text{task}}(T, x, \alpha, h) = \int_0^T \underbrace{L\left(\mathbb{E}x(t), \alpha(t), h(t)\right)}_{\text{incremental cost}} dt + \underbrace{V[\mathbb{E}x(T), \alpha(T)]}_{\text{terminal cost}} \tag{3.1}$$

where L and V respectively denote the incremental and terminal cost models associated with the task (following the notation of Ref [72]). Using this definition, allowing $T \to \infty$ creates a persistent task while setting $L = 0$ creates a task with only a terminal cost/reward. The model of incremental costs is a potential source of errors and is a function of the expected value of the state at time t, $\mathbb{E}x(t)$, agent roles $\alpha(t)$ and the disturbed control input $h(t) = u(t) + w(t)$, with disturbance $w(t)$. In this representation, both $x(t)$ and $w(t)$ are uncertain. To simplify the task allocation optimization (via reduction of the search space), we assume task independence: task cost is not a function of any other task cost and no ordering constraints are placed on any set of tasks. That is, each task can be evaluated and assigned independent of any others. This is not to say that we cannot require specific actions to follow a prescribed sequence. For such a case, we simply lump the constrained set of actions together into a single task.

Regardless of how it is formulated (e.g. MILP [2, 5] MDP [20], CBBA [26]), the cooperative planner, or cooperative control algorithm, is the source for baseline plan generation within *iCCA*. It is important to note that the performance and robustness properties of the integrated system rely on the accuracy of the planner. In other words, we assume contains sufficient models to avoid costly mistakes, but is not necessarily detailed enough to provide an optimal solution. Therefore, the planner in *iCCA* is augmented to provide access to these internal models so that the learning element can assist in their online refinement. For those planner types that do not maintain and/or use explicit internal models, the learner is either embedded into the planner (e.g. reinforcement learning based planners), or the learner is implemented to alter planner output under certain conditions. An example of this type of planner-learner relationship is shown in Section 3.4.3, where the learner suggests candidate actions to the planner.

Output from the performance analysis element is also available to the planner for use in its internal optimization. In general, the cooperative planning algorithm can act directly upon the performance observed. For example, measured versus expected performance can produce what is often referred to as *temporal-difference errors* [17], which can drive a multitude of objective functions, including those found in many cooperative planning algorithms. In effect, we connect the cooperative planner with both a learning method and a performance analysis method in an attempt to generate cooperative control solutions that are both robust and adaptable to errors and uncertainties without being unnecessarily conservative.

Building on the above definition of "task", we consider first the generalized assignment problem, which underlies many multi-agent task allocation schemes, including the problem of interest: *cooperative tasking*. We then introduce the cooperative tasking, or cooperative planning problem, and formulate it as both a dynamic program and as a decentralized auction.

3.1.1 Generalized Assignment Problem Statement

The multi-agent task allocation problem is, in essence, an instance of the generalized assignment problem, which is an NP-hard combinatorial optimization even when omitting the possibility of uncertainty. It can be stated as follows: *Consider N_A agents and N_T tasks where any agent can be assigned to perform any task. Each task has both incremental and terminal costs/rewards that may vary depending on the agent-task assignment. Find an overall assignment that minimizes the total cost J while respecting each agent's budget w_i.* The general problem can formally be written as the following optimization:

$$\begin{aligned}
\text{minimize} \quad & J = \sum_{i=1}^{N_A}\sum_{j=1}^{N_T} x_{ij} J_{ij} \qquad (3.2)\\
\text{s.t.} \quad & \sum_{j=1}^{N_T} x_{ij} J_{ij} \leq w_i, \; i = 1, \dots N_A \\
& \sum_{i=1}^{N_A} x_{ij} \leq 1, \; j = 1, \dots N_T \\
& x_{ij} \in \{0, 1\}, \; i = 1, \dots N_A, \; j = 1, \dots N_T
\end{aligned}$$

where x_{ij} are the binary decision variables indicating whether agent i is assigned to task j and J_{ij} represents the cost of such an assignment, which can be expressed as in (3.1) through choice of incremental and terminal costs for task j. The first constraint ensures that each agent does not incur more cost than the designated budget w_i allows. The second constraint ensures that each task is assigned to a maximum of one agent, and the final constraint enforces the binary nature of each agent-task assignment. Under this formulation, the optimization is centralized and coordination is forced. In addition, the optimization in (3.2) must be re-evaluated whenever there is a change in the number of agents N_A, the number of tasks N_T, or in the cost structure J_{ij}.

3.1.2 Cooperative Planner Problem Statement

Cooperative planning is rooted in the generalized assignment problem and can be expressed with almost identical language as follows: *Consider N_A agents and N_T*

tasks where agent and task types must be compatible for assignment. Each task has models for both incremental and terminal costs/rewards that may vary depending on the agent-task assignment. Find an overall assignment that attempts to minimize the total cost J while respecting each agent's budget w_i. The cooperative planning problem can be formulated in many ways, some of which are more natural than others given specifics of the application. For example, agent models (e.g. dynamics, health, behavior, rules-of-engagement) may have a significant effect on the optimality and sensitivity of plan output, yet are omitted from many problem formulations, as in the generalized assignment problem. Formulations such as dynamic programming however, can capture these models at the price of added complexity. In the subsections that follow, we outline two formulations of interest: dynamic programming and decentralized auctions.

Cooperative Planning via Dynamic Programming

Dynamic programming (DP) is a powerful framework for solving sequential decision-making problems, such as multi-agent task allocation. The DP approach consists of two main components: a model of system dynamics and a cost function [12, 23]. The dynamic model describes the evolution of the state of the system over time under a given sequence of actions and is susceptible to modeling errors and parametric uncertainties. The cost function is additive over time and depends on the state of the system. Solving a DP amounts to finding a state-dependent rule for action selection, $\pi^\star$, that minimizes the total expected time-discounted cost incurred, as given by

$$J_{\pi^\star}(s_0) = \min_{\pi \in \Pi} \mathbb{E} \sum_{k=0}^{\infty} \alpha^k g\left(s_k, \pi(s_k)\right), \tag{3.3}$$

where the expectation $\mathbb{E}$ is taken over all possible future states $\{s_1,\ s_2,\ \ldots\}$, given the initial state s_0 and the policy π. This policy can be extremely sensitive to variations in the dynamic model, including mild uncertainties in the model parameters [86]. This motivates the desire to learn model parameters over time.

We begin the DP formulation of the multi-agent task allocation problem by defining the system "state" at any given stage k by the number of tasks remaining to be allocated, $n_T(k)$, and the number of agents available for assignment, $n_A(k)$. Letting $x_k = [n_T(k),\ n_A(k)]^T$ denote a state in the state space $\mathcal{S} = \mathbb{Z}_+ \times \mathbb{Z}_+$, one possible DP formulation is due to Alighanbari [2] and is given by the following Bellman equation:

$$\begin{aligned} J_k(x_k) &= \min_{u_k, |u_k| \leq n_A(k)} \left(\sum_{i \in u_k} P_i(k) v_i + \alpha J^\star_{k+1}(x_{k+1}) \right) \ \forall x_k \in \mathcal{S} \\ \text{s.t.} \quad & x_{k+1} = [n_T(k) - |u_k|,\ n_A(k) - |u_k|]^T \\ & k \in \{0, ..., K-1\} \\ & J^\star_K = 0 \end{aligned} \tag{3.4}$$

where u_k denotes the set of tasks to be allocated at time k, v_i is the value of task i in the set u_k, P_i is the probability of successfully performing task i and K is the maximum number of timesteps considered. Planner performance at time k is encoded in J_k. In this formulation, $P_i(k)$ and v_i are potential sources of modeling errors and may encode significant uncertainty. In addition, computation time grows exponentially in the number of tasks, making this approach infeasible for large teams of agents and/or large numbers of tasks, and therefore ripe for new approaches toward approximations and learned simplifications, such as those presented in Chapter 4.

Cooperative Planning via Decentralized Auction

The Consensus-Based Bundle Algorithm (CBBA) [26] is a decentralized auction-based approximation to the generalized assignment problem given in Section 3.1.1. The objective of CBBA is to find a conflict-free assignment of tasks to agents that maximizes some global reward. An assignment is said to be free of conflicts if each task is assigned to no more than one agent. The global objective function is assumed to be the sum of local reward values, while each local reward is determined as a function of the tasks assigned to that particular agent. CBBA consists of iterations between two phases: a bundle building phase where each agent greedily generates an ordered bundle of tasks, and a consensus phase where conflicting assignments are identified and resolved through local communication between connected agents.

While the details of the formulation are left to Ref [26], there are several core features of CBBA that can be utilized to develop an efficient planning mechanism for heterogeneous teams. First, CBBA is a decentralized decision architecture, resulting in a lower computational overhead as required for centralized planning with a large number of agents. Second, CBBA is a polynomial-time algorithm [26]. The worst-case complexity of the bundle construction is $\mathcal{O}(N_T L_t)$, where N_T is the number of tasks and L_t is the upper-bound on the length of an agent's task bundle ($L_t \leq N_T$). CBBA converges within $\max\{N_T, L_t N_A\}D$ iterations, where D is the network diameter ($D < N_A$). Thus, the CBBA framework scales well with the size of the network and/or the number of tasks (or equivalently, the length of the planning horizon). Third, various design objectives, agent models, and constraints can be incorporated by defining appropriate scoring functions. If the resulting scoring scheme satisfies a certain property called *diminishing marginal gain* (DMG) [26], a provably good feasible solution is guaranteed. The application of learning algorithms within the scoring function can yield significant improvements in planner performance.

3.2 Performance Analysis

One of the main reasons for encouraging cooperation in a MAS is to minimize some cost, or otherwise optimize some objective function. Very often this objective involves time, risk, fuel, or other similar physically-meaningful quantities. The purpose of the performance analysis module is to accumulate observations, glean useful information buried in the noisy observations, categorize it and use it to improve subsequent plans. In other words, the performance analysis element of iCCA attempts to improve agent behavior by diligently studying its own experiences [91] and compiling relevant signals to drive the learner and/or the planner.

The use of such feedback within a planner is of course not new. In fact, there are very few cooperative planners which do not employ some form of measured feedback. The focus of the performance analysis block within *iCCA* is to extract relevant information from the observation stream and formulate a meaningful metric that can be used in the planner itself, and/or as input to the learner. What exactly is, or can be, extracted from the observations is, of course, application specific. However, as the objective is performance *improvement*, this module is connected with the cooperative planner so as to gain access to the underlying cost/reward function of the problem.

In Section 3.4, we implement several methods of performance analysis based on observed data. In the first example, we construct temporal-difference errors based on expected and observed cost and use these errors to drive the learning of uncertain parameters. Second, we record state-dependent observations to construct the parameters used by the learner. Finally, we implement a method for analyzing risk as a form of performance and couple this with a reinforcement learning algorithm.

3.3 Learner

Although learning has many forms, iCCA provides a minimally restrictive framework where the contributions of the learner include the following:

- Assist the cooperative planner by adapting to parametric uncertainty of internal models
- Suggesting candidate actions to the analysis module that the learner sees as beneficial

Learning can leverage the multi-agent setting by collecting observations from each member of the team and using the information from sensor data and observed or inferred mission successes (and failures) as feedback signals to identify possible improvements, such as tuning the weights of an objective function. However, a trademark of learning algorithms is that negative information is extremely useful, albeit extremely costly in the cooperative control setting. Active learning algorithms can explicitly balance the cost of information gathering against the expected value of information gathered. In the sections that follow, several types of learning algorithms are considered for inclusion in the iCCA framework, namely reinforcement learning, maximul-likelihood estimation, and adaptive control techniques.

3.3.1 Reinforcement Learning

The underlying goal of the two reinforcement learning algorithms presented here is to improve performance of the cooperative planning system over time using observed rewards by exploring new agent behaviors that may lead to more favorable outcomes. The details of how these algorithms accomplish this goal are discussed in the following sections.

Sarsa

A popular approach among MDP solvers is to find an approximation to $Q^{\pi}(s,a)$ (policy evaluation) and update the policy with respect to the resulting values (policy improvement). Temporal Difference learning (TD) [103] is a traditional policy evaluation method in which the current $Q(s,a)$ is adjusted based on the difference between the current estimate of Q and a better approximation formed by the actual observed reward and the estimated value of the following state. Given $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$ and the current value estimates, the temporal difference (TD) error, δ_t, is calculated as:

$$\delta_t(Q) = r_t + \gamma Q^{\pi}(s_{t+1}, a_{t+1}) - Q^{\pi}(s_t, a_t).$$

The one-step TD algorithm, also known as TD(0), updates the value estimates using:

$$Q^{\pi}(s_t, a_t) = Q^{\pi}(s_t, a_t) + \alpha \delta_t(Q), \tag{3.5}$$

where α is the learning rate. Sarsa (state action reward state action) [102] is basic TD for which the policy is directly derived from the Q values as:

$$\pi^{Sarsa}(s,a) = \begin{cases} 1-\epsilon, & a = \text{argmax}_a Q(s,a) \\ \frac{\epsilon}{|\mathcal{A}|}, & \text{Otherwise} \end{cases}.$$

This policy is also known as the ϵ-greedy policy[1]. It is important to note however, that many exploration strategies can be used here, in place of ϵ-greedy.

Natural Actor-Critic

In reinforcement learning, a major challenge is the so-called "exploration vs. exploitation" trade-off. Often, reinforcement learning algorithms use an ϵ-greedy approach to dictate their exploration. This essentially means that with a fixed-probability ϵ, the agent will choose to explore by randomly choosing an action. Otherwise, with probability $1-\epsilon$, the agent will greedily choose the action that most likely leads to the state with the highest value or, in other words, the action with the highest Q-value. This approach is simple to implement and has some nice properties (e.g. it yields ergodic policies), but consider if the probability of exploration were adjustable based on past experience and observations? It is conceivable that this would improve exploration. One method to accomplish this is to separate the policy and the Q-function into different elements. Actor-critic methods parameterize the policy and store it as a separate entity named the *actor*. In this thesis, the actor is a class of policies represented as the Gibbs softmax distribution:

$$\pi^{AC}(s,a) = \frac{e^{P(s,a)/\tau}}{\sum_b e^{P(s,b)/\tau}},$$

in which $P(s,a) \in \mathcal{R}$ is the preference of taking action a in state s, and $\tau \in [0,\infty)$ is a "temperature" knob allowing for shifts between greedy and random action selection. For a tabular representation, the actor update amounts to:

$$P(s,a) \leftarrow P(s,a) + \alpha Q(s,a)$$

following the incremental natural actor-critic framework [24]. The value of each state-action pair ($Q(s,a)$) is held by the *critic* and is calculated/updated in an identical manner to Sarsa in Equation (3.5).

[1]Ties are broken randomly, if more than one action maximizes $Q(s,a)$.

3.3.2 Maximum-Likelihood Estimation

Maximum likelihood estimation is an analytic procedure to obtain the most likely estimate of a parameter that produced, or is otherwise relevant to, a set of data samples. In the context of the iCCA framework, the parameters of interest are typically internal to the Cooperative Planning module (e.g. how fast the agent can move, the probability of a certain event occuring, etc.) and the data samples are the result of observations and are provided by the Performance Analysis module. The process of calculating the maximum likelihood estimate begins with writing out the *likelihood function* of the data samples, i.e. the probability of obtaining these samples given some model of their underlying probability distribution. This model is not perfectly known, and contains unknown and/or uncertain parameters. The values of these parameters that maximize the sample likelihood are known as the Maximum Likelihood Estimates or MLE's [50].

Drawbacks to maximum likelihood estimation include the fact that the resulting MLE can be heavily biased when only a small set of samples is used, and that calculating the MLE can require solving very complex non-linear equations. However, on the positive side, the MLE becomes an unbiased, minimum variance estimate as the sample size increases and MLEs have approximate normal distributions and approximate sample variances that can be used to generate confidence bounds [50].

3.3.3 Adaptive Control

Adaptive control is an approach for handling parameter uncertainties and/or time-varying systems. Application examples include fire-fighting helicopters that experience drastic changes in mass during a mission or high-performance aircraft that are capable of flying in sub- and super-sonic flight envelopes [64]. In these cases, parameters that influence system dynamics are changing over time, or are not precisely known. Adaptive control is an ideal candidate for such systems since online parameter estimation is integrated into the control scheme, thus enabling consistent performance even in the presence of model-parameter variations and uncertainties.

In Section 3.4, we give examples of active and passive maximum likelihood learners in the context of a multi-agent Markov Decision Processes and a decentralized market-based planner. Finally, we give an example of an actor-critic reinforcement learner [24] biased and guided in its exploration by a decentralized market-based planner.

3.4 Example Implementations

In this section, we implement *iCCA* in the context of several cooperative control scenarios and show how mission performance is improved over more traditional cooperative control strategies. First, we use a decentralized auction-based algorithm called consensus-based bundle algorithm (CBBA) [26] as the cooperative planner. Second, we implement a multi-agent Markov decision process (MMDP) with uncertain model parameters. Finally, we again use a CBBA planner, but with an actor-critic reinforcement learning algorithm and a performance analysis module based on the notion of risk. Aside from the performance improvements, another result of these example problems is the realization that the planning algorithm needs to be solvable on a timescale that is consistent with the planning frequency and planning horizon of the problem. In some cases, with the MDP-based planners in particular, this online-solvable characteristic presents a real challenge — one that is addressed in Chapter 4.

3.4.1 Consensus-Based Cooperative Task Allocation

In this example, we consider a multi-agent task-allocation scenario where each agent has a specific set of capabilities, and each task a set of requirements. The challenge is to find the assignment of agents to tasks that minimizes the objective function while satisfiying compatability and other constraints such as specific time-windows when the tasks need to be completed. CBBA (see 2.2.1) was implemented as the baseline cooperative planner with several extensions to account for agent-task compatibility and expected fuel consumption during task execution. In order to account for a heterogeneous team, agents were classified according to their capabilities and tasks according to their requirements. A set of constraints were then be incorporated into the planning process specifying which types of agents can do which types of tasks (i.e. UAVs can perform aerial surveillance, ground teams can perform rescue operations, etc). As formulated in Ref [82], one straightforward way to incorporate these constraints is by enforcing agents to bid *zero* for tasks with which they are not compatible.

To account for fuel consumption, the score function was augmented with a fuel penalty due to travel distance,

$$c_j(\tau_{ij}^\star) = e^{-\lambda_j(\tau_{ij}^\star - t_{j_{start}})} R_j u_j(\tau_{ij}^\star) - F_i \Delta D_{ij}(\mathbf{p}_i)$$

where F_i is the cost of fuel per meter incurred by agent i and $\Delta D_{ij}(\mathbf{p}_i)$ is the distance traveled by the agent to get to the task location from its previous location. To ensure satisfaction of DMG, a heuristic distance, $\Delta D'_{ij}$, representing the distance from the vehicle's *initial position* to the task location, was used instead. Monte Carlo simulation results verified that this type of heuristic penalty produced equally efficient task allocation assignments as those obtained using the actual travel distance $\Delta D_{ij}(\mathbf{p}_i)$, while guaranteeing convergence of the algorithm to a conflict-free assignment [82].

Implementation Under iCCA

Figure 3-2 shows how this example was wrapped within the framework of *iCCA*. As seen, the CBBA algorithm serves as the cooperative planner which uses internal models of vehicle dynamics to rank bids placed by participating agents. These models, like many used in cooperative control algorithms, are good approximations and serve their purpose well. However, refining these model parameters online can increase overall performance without sacrificing robustness. As an example, one can reasonably expect the outcome of the planner to be sensitive to variations in the agent' estimated cruise velocity, as this parameter is used to calculate every bid an agent places. These variations could easily be due to external conditions such as wind, agent congestion, collision avoidance routines, etc. Over- or under-estimating

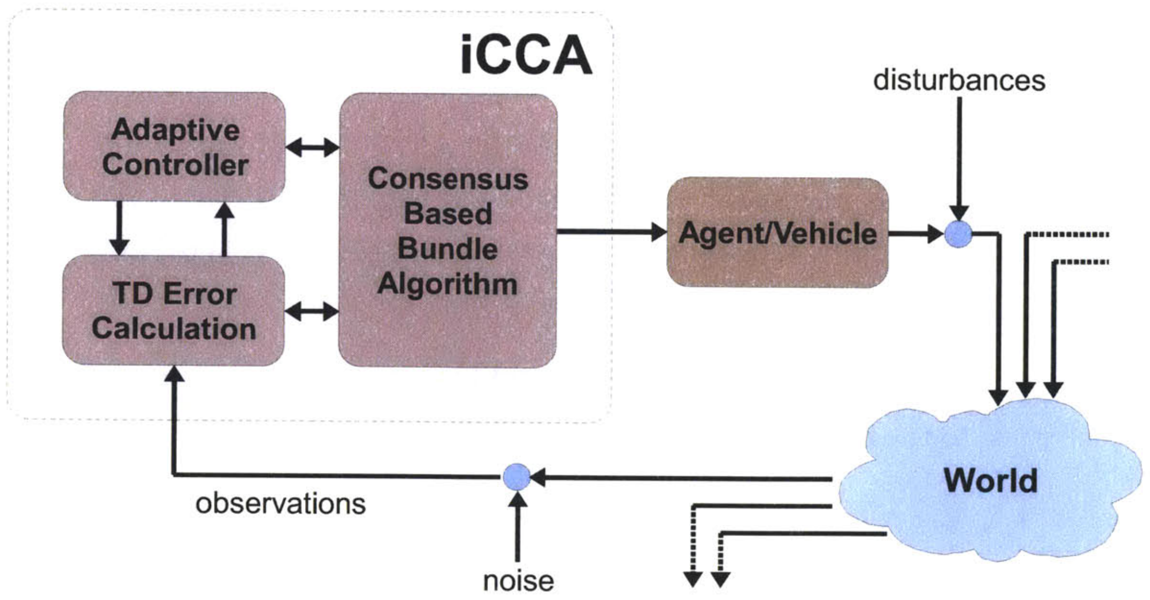


Figure 3-2: *iCCA* formulation with a CBBA planner and a simple learner driven by temporal-difference errors

its cruise velocity causes the agent to underperform with respect to its expected payoff as penalties are incurred for arriving outside the specific time-window for a task. Therefore, if the agent could improve the estimate of its cruise velocity over time, it would yield a lower-cost solution to the task-assignment problem.

In response to this, we implemented a simple, direct-adaptive controller as the learning element of this instance of *iCCA* with inputs from the performance analysis module in the form of temporal-difference errors. In direct-adaptive control, the plant model is re-parameterized in terms of the controller parameters and then an estimator is run on this model - thus, the controller parameters are estimated directly. In such an approach, the gradient algorithm is commonly used to generate the adaptation rule for evolving these parameters such that they converge to their actual values. In general, the gradient rule can be written as:

$$\dot{\theta}_1 = -\gamma \left(y_p - y_m\right) r$$

where y_p and y_m are the plant and model parameters respectively and their difference constitutes a temporal-difference error.

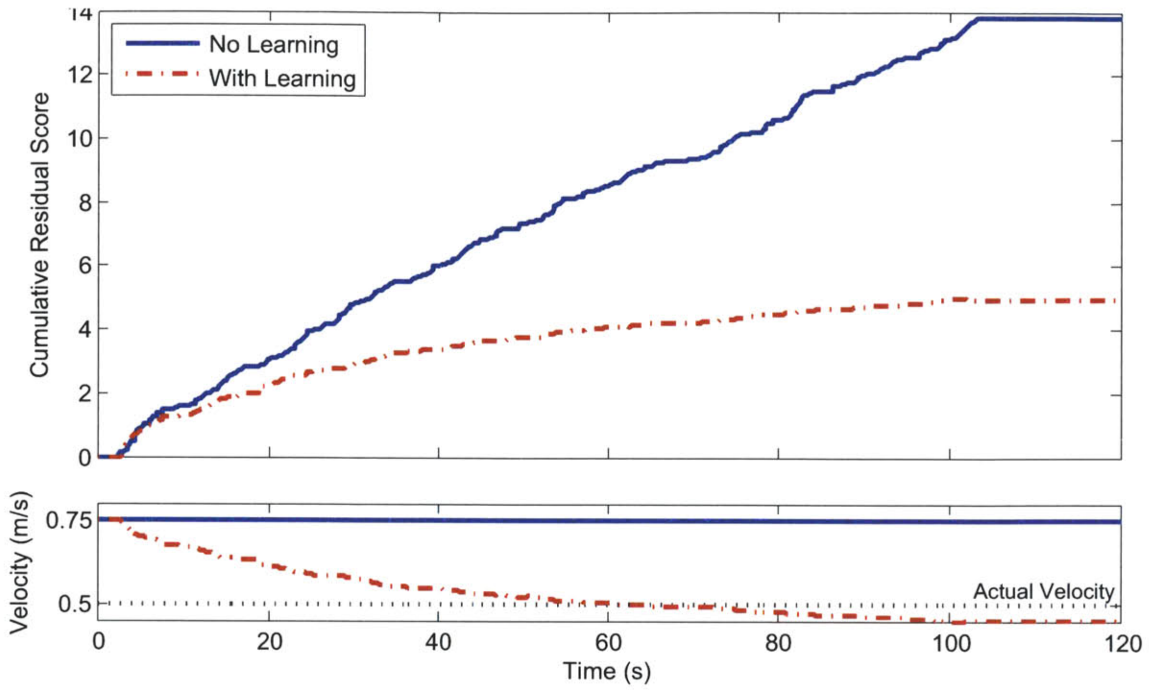


Figure 3-3: Cumulative residual score (expected - actual) over the duration of the mission. When coupled with a learner, the planner more accurately anticipates performance.

The performance analysis element, labeled "TD Error Calculation", observes and collects the scores achieved while servicing the tasks won during the bidding process. It then compares these actual with expected scores received from queries directly to the CBBA planner. A temporal-difference (TD) error of the form $\delta = E[\underline{x}] - \underline{x}$ captures the discrepancy between the two scores and is used to drive a direct adaptive controller which learns the true cost of fuel. In the simulated mission scenario, a time-discounted reward is received as a result of accomplishing a 5 s task within the allowable time window of 15 s, while costs are accrued as a result of travel.

Figure 3-3 shows the residual mission score as a function of time. As expected, the performance of the planning scheme increases when coupled with a learning algorithm to reduce the uncertainty around nominal cruise velocity. This enables agents to place more accurate bids and collect actual scores that are much closer to their expected scores.

3.4.2 Multi-agent Persistent Surveillance

In this example, we formulate a multi-agent Markov Decision Process (MMDP) while considering a persistent surveillance mission scenario as outlined in Ref [21]. MDPs are a natural framework for solving multi-agent planning problems as their versatility allows modeling of stochastic system dynamics as well as inter-dependencies between agents [65, 84]. In the persistent surveillance problem, a group of N_a UAVs are each equipped with some type(s) of sensors and are initially located at a base location. The base is separated by some (possibly large) distance from the surveillance location and the objective of the problem is to maintain a specified number N_r of requested UAVs over the surveillance location at all times while minimizing fuel costs. This represents a practical scenario that can show well the benefits of agent cooperation.

The uncertainty in this case is a simple fuel consumption model based on the probability of a vehicle burning fuel at the nominal rate, p_f. That is, with probability p_f, vehicle i will burn fuel at the known nominal rate and with probability $1 - p_f$, vehicle i will burn fuel at twice the known nominal rate. When p_f is known exactly, a policy can be constructed to optimally hedge against running out of fuel while maximizing surveillance time and minimizing fuel consumption. Otherwise, policies constructed under overly conservative (p_f too high) or naive (p_f too low) estimates of p_f will respectively result in vehicles more frequently running out of fuel, or a higher frequency of vehicle phasing (which translates to unnecessarily high fuel consumption). Given this qualitative statement of the persistent surveillance problem, an MDP is formulated as detailed in Ref [86].

Implementation under iCCA

Wrapping the above problem statement within *iCCA*, the multi-agent MDP becomes the cooperative planner, while the performance analysis block consists of a state-dependent observation counter that tracks discretely observed fuel-burn events. Two learning algorithms were implemented in this example scenario: an active and a passive learner. Both of which are based on a maximum likelihood algorithm whose

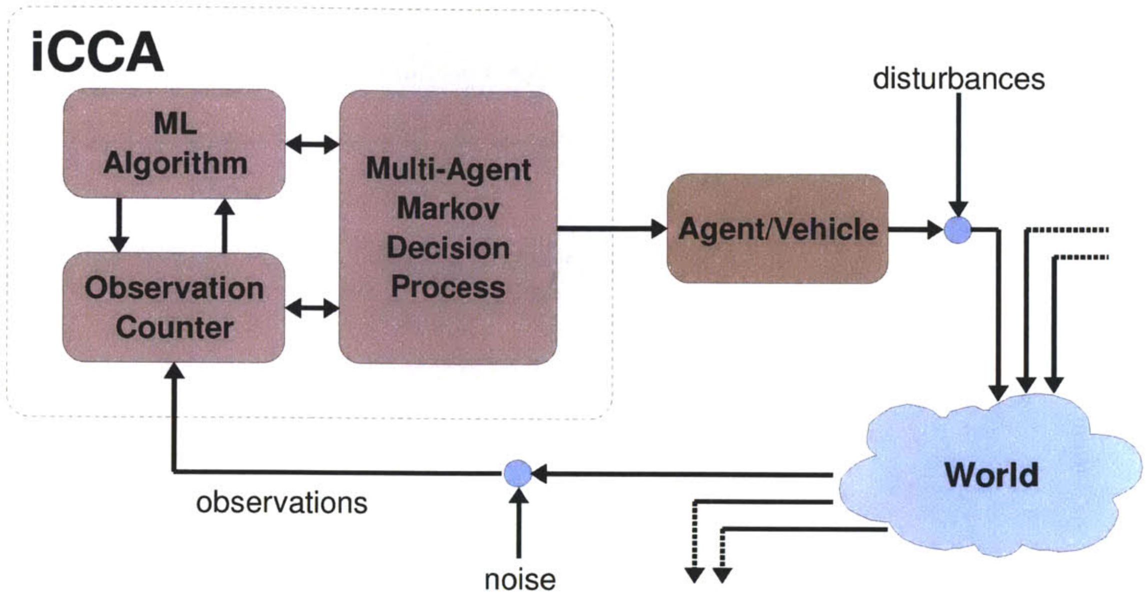


Figure 3-4: *iCCA* formulation with a multi-agent MDP planner and a ML learner driven by β-distribution observation counts.

parameters are modeled as a β-distribution and are updated using the number of observed fuel burn events, α_i, as counted by the performance analysis element, as depicted in Figure 3-4.

First, the passive learner simply uses these α_i inputs to calculate an estimate of the probability of nominal fuel burn, $\hat{p}_f$, and its corresponding variance. Second, the active learner which, after calculating $\hat{p}_f$ and the corresponding variance, searches the possible actions and "suggests" to the planner that it take the action leading to the largest reduction in variance around p_f.

For the case of the active learner, we embed a maximum-likelihood (ML) estimator into the MDP formulation such that the resulting policy will bias exploration toward those state transitions that will result in the largest reduction in the expected variance of the ML estimate $\hat{p}_f$. The resulting cost function is then formed as

$$g'(\mathbf{x}, \mathbf{u}) = g(\mathbf{x}, \mathbf{u}) + C\sigma^2(\widehat{p}_f)(\mathbf{x}) \tag{3.6}$$

where C represents a scalar gain that acts as a knob we can turn to weight exploration, and $\sigma^2(\widehat{p}_f)(\mathbf{x})$ denotes the variance of the model estimate in state $\mathbf{x}$. The variance of

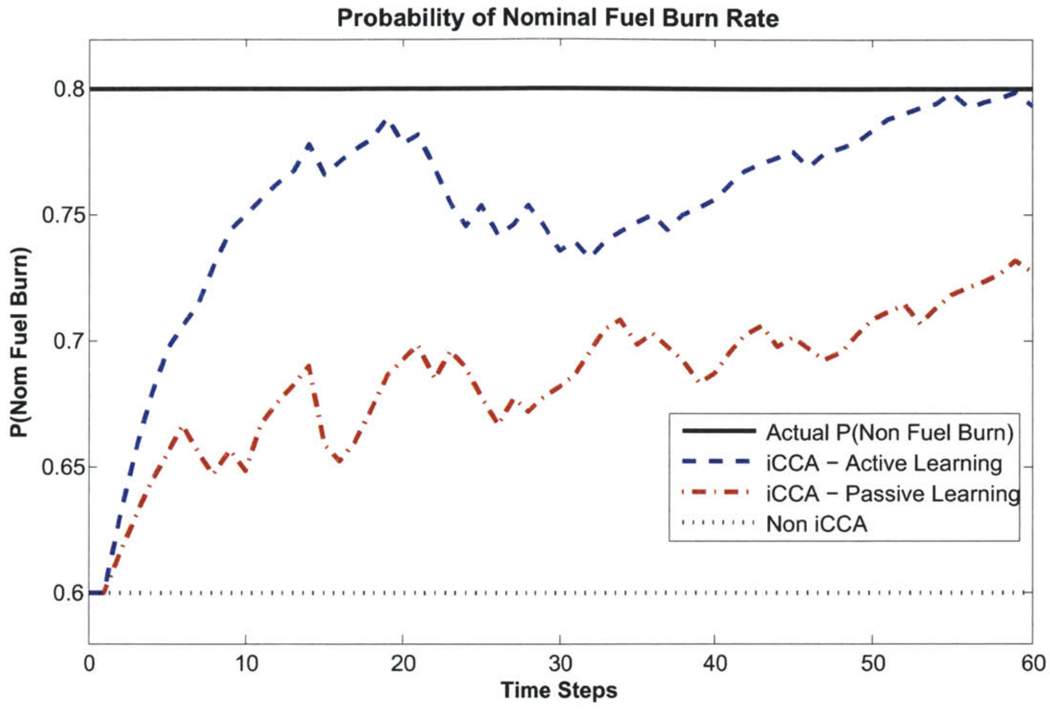


Figure 3-5: Comparison of the fuel burn model parameter learned over time under the MDP formulation

the Beta distribution is expressed as

$$\sigma^2(\widehat{p}_f)(\mathbf{x}) = \frac{\alpha_1(\mathbf{x})\alpha_2(\mathbf{x})}{(\alpha_1(\mathbf{x}) + \alpha_2(\mathbf{x}))^2(\alpha_1(\mathbf{x}) + \alpha_2(\mathbf{x}) + 1)} \tag{3.7}$$

where $\alpha_1(\mathbf{x})$ and $\alpha_2(\mathbf{x})$ denote the counts of nominal and off-nominal fuel flow transitions observed in state $\mathbf{x}$ respectively, by the performance module labeled "Observation Counter" in Figure 3-4.

Figure 3-5 compares the rate at which the model parameter p_f is learned by the ML estimator that uses online observations of vehicle fuel consumption. In the passive learning case, the planner chooses actions based on an internal objective function, without being "nudged" by the learner. These actions lead to observations, which in turn reduced the variance around p_f, albeit not as quickly as the active learner, as seen in Figure 3-5. For the case without *iCCA*, no learning is achieved and the planner assumes p_f is known, when in fact it is not, and therefore acts sub-optimally

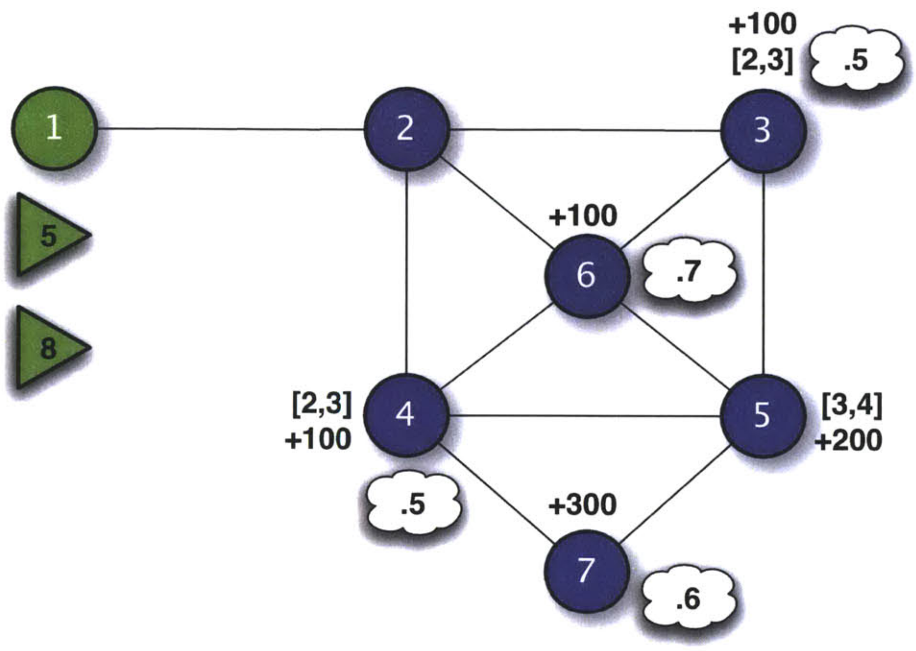


Figure 3-6: A team of two UAVs (triangles) maximize their reward by cooperating to visit targets. Target nodes (circles) have a probability of receiving a reward (positive values with probability in nearby cloud). Note that some target nodes have no value. Allowable visit times shown in brackets.

e.g. running a higher risk of crashing, or wasting fuel.

3.4.3 Actor-Critic Policy Learning

In this example, we integrate a variant of the CBBA planner presented in Section 3.4.1 with an actor-critic reinforcement learner in the context of a stochastic multi-agent task allocation scenario. We discuss a method for learning and adapting cooperative control strategies in stochastic domains. We consider a small team of agents that start by following a "safe" plan, as calculated by the baseline planner and incrementally adapting it to maximize rewards by cooperating to visit target nodes in the network.

Figure 3-6 depicts the problem scenario where the base is highlighted as node 1 (green circle), targets are shown as blue circles and agents as triangles. The total amount of fuel for each agent is highlighted by the number inside each triangle. For those targets with an associated reward it is given a positive number nearby. The

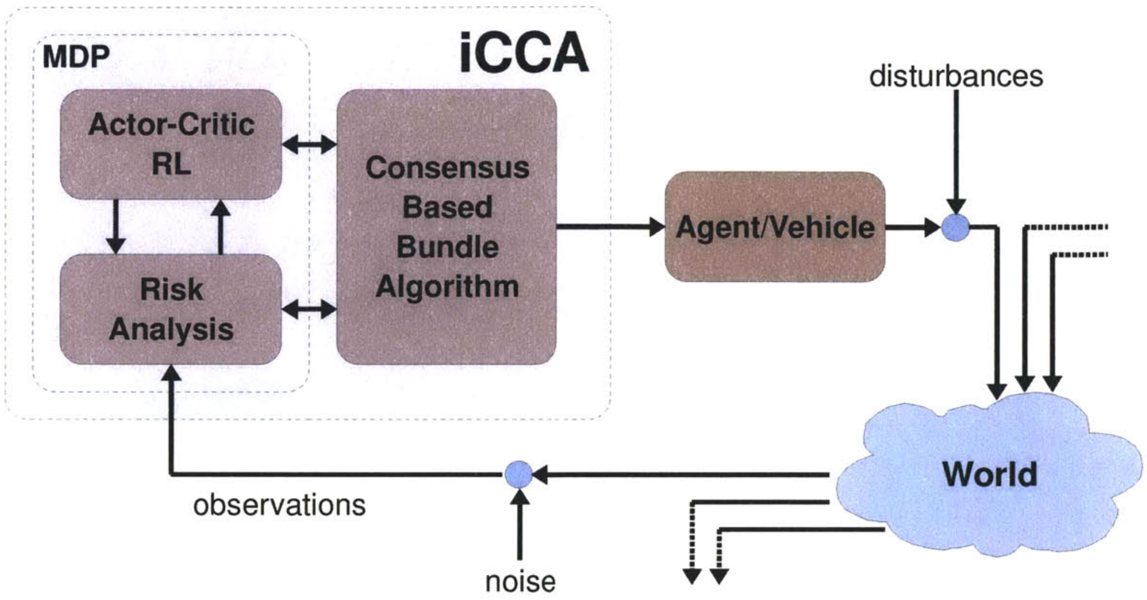


Figure 3-7: *iCCA* framework as implemented. CBBA planner with the risk analysis and actor-critic learner formulated under an MDP.

constraints on when the target can be visited are given in square brackets and the probability of receiving the known reward when the target is visited is given in the white cloud nearest the node.[2] Each reward can be obtained only once and all edges require one unit of fuel and one time step to traverse. We also allow UAVs to loiter on any nodes for the next time step. The fuel burn for loitering action is also one except for the UAVs staying in the base, where they are assumed to be stationary and sustain no fuel cost. The mission horizon was set to 8 time steps.

Implementation under iCCA

We implemented the consensus-based bundle algorithm under the same formulation as outlined in Section 3.4.1 with additional fuel constraints. Specifically, each agent was given a limited amount of fuel and the combined path length of its task bundle was constrained to be feasible with respect to fuel consumption while allowing the agent enough reserve to return to base upon completion.

For the learning algorithm, we implemented an actor-critic reinforcement learner

[2]If two agents visit a node at the same time, the probability of visiting the node would increase accordingly.

[24] which uses information regarding performance to explore and suggest new behaviors that would likely lead to more favorable outcomes than the current behavior would produce. In actor-critic learning, the actor handles the policy, where in our experiments actions are selected based on Gibbs softmax method:

$$\pi(s,a) = \frac{e^{P(s,a)/\tau}}{\sum_b e^{P(s,b)/\tau}},$$

in which $P(s,a)$ is the preference of taking action a in state s, and $\tau \in (0,\infty]$ is the temperature parameter acting as a knob shifting from greedy towards random action selection. Since we use a tabular representation, the actor update amounts to $P(s,a) \leftarrow P(s,a) + \alpha Q(s,a)$, where α is the learning rate.

As for the critic, we employ the temporal-difference learning algorithm [102] to update the associated value function estimate. We initialized the actor's policy by bootstrapping the initial preference of the actions generated by CBBA.

The performance analysis block is implemented as a "Risk Analysis" tool where actions suggested by the learner can be overridden by the baseline cooperative planner if they are deemed too "risky". This synergistic relationship yields a "safe" policy in the eyes of the planner, upon which the learner can only improve. A trademark of learning algorithms in general, is that negative information is extremely useful in terms of the value of information it provides. This is unacceptable in a multi-agent environment. We therefore introduce the notion of a "virtual reward" - a large negative value delivered to the learner for suggesting risky actions. When delivered, the learner associates this negative reward with the previously suggested action, dissuading the learner from suggesting it again.

The formulation shown in Figure 3-7 allows for the key concept of *biased* and *guided* exploration such that the learner can explore the parts of the world that are likely to lead to better system performance while ensuring that the agent remain safely within its operational envelope and away from states that are known to be undesirable, such as running out of fuel. The *bias* is introduced as the initial policy is seeded using plans generated by the baseline CBBA planner. In order to facilitate

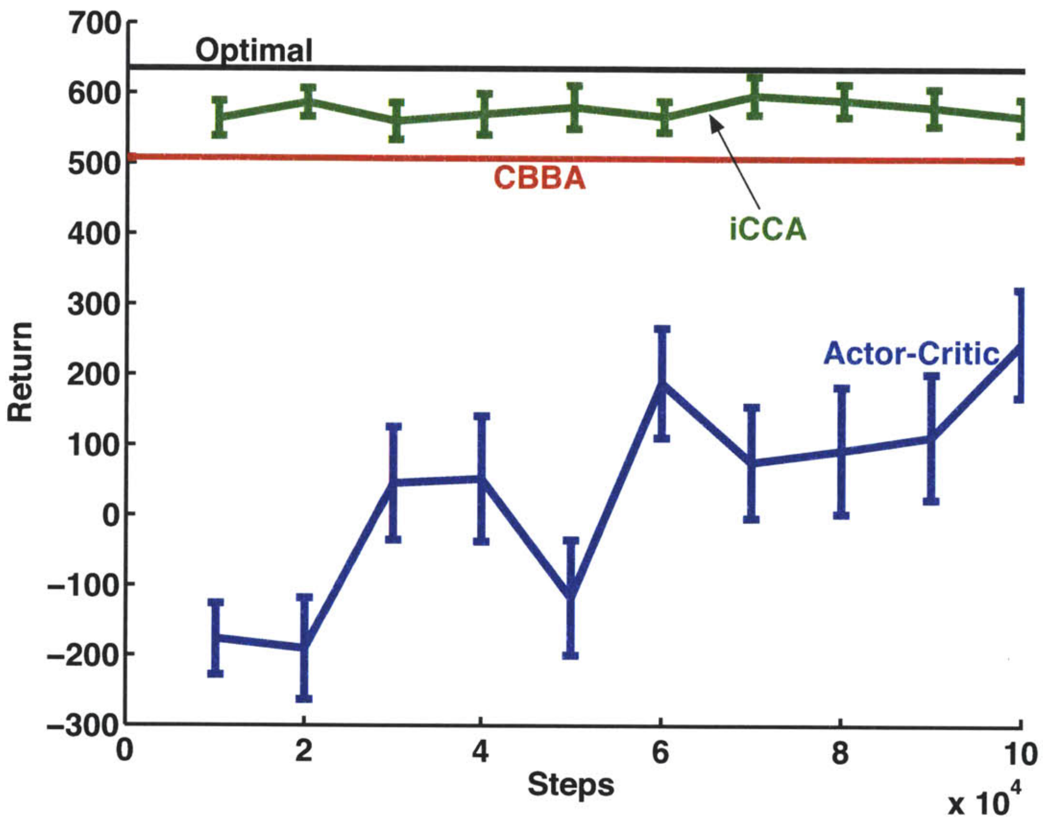


Figure 3-8: A comparison of the collective rewards received when strictly following plans generated by CBBA alone, actor-critic reinforcement learning inside/outside of the iCCA environment. All are compared against the optimal performance as calculated via dynamic programming.

the *bound*, the risk analysis module inspects all actions suggested by the actor and replaces the "risky" ones with the action specified by CBBA, thus guiding the learning away from catastrophic errors. In essence, the baseline cooperative control solution provides a form of "prior" over the learner's policy space while also acting as a backup policy in the case of an emergency.

We first solved the problem using backward dynamic programing in order to use this solution as a benchmark for comparison (this took about a day and cannot be easily scaled for larger sizes of the problem). We then ran CBBA on the expected deterministic problem (as converted from the stochastic problem), and ran it for 10,000 episodes. For all experiments, we set the preference of the advised CBBA state-action pairs to 100. τ was set to 1 for the actor. Figure 3-8 depicts the performance of iCCA and Actor-Critic averaged over 60 runs. The Y-axis shows the cumulative reward, while the X-axis represents the number of interactions. Each point on the graph is the result of running the greedy policy with respect to the existing preferences of the actor. For iCCA, risky moves again were replaced by the CBBA baseline solution. Error bars represent the standard error with 90% confidence interval. In order to show the relative performance of these methods with offline techniques, the optimal and CBBA solutions are highlighted as lines. It is clear that the actor-critic performs much better when wrapped into the iCCA framework and performs better than CBBA alone. The reason is that CBBA provides a good starting point for the actor-critic to explore the state space, while the risk analyzer filters risky actions of the actor which leads into catastrophic scenarios.

3.5 Summary of iCCA

In summary, this chapter presented an archticture designed for the tight integration of planning and learning techniques so as to enable performance increases as uncertain parameters are learned and/or as an agent's policy is improved over time. The iCCA framework consists of four modules: an autonomous agent, a cooperative planner, a performance analysis module and a learning module. Working together in a single environment, these modules enable a more intelligent behavior in the autonomous agent as observations accrue and as parameter uncertainty is reduced. However, through the example implementations it became clear that to gain full advantage of the learning and performance analysis modules within iCCA, the planner should be solvable on a timescale that is congruent with the learning rate. As a result of this, the following chapter develops several approaches for approximating the cooperative planner to enable faster solution times.

Chapter 4

Approximate Decentralized Multi-Agent Planning

In the previous chapter, the intelligent cooperative control architecture was developed to enable the tight integration of planning and learning methods within cooperative multi-agent systems. However, through implemented examples, it became clear that for a cooperative planning algorithm to take full advantage of online learning and adaptation techniques, it needed to be solvable on a much shorter timescale than many current formulations allow. Therefore, the purpose of this chapter is to address this problem by developing an approximation approach that reduces the complexity of multi-agent Markov decision processes (MMDPs) through targeted approximations and decentralization.

This chapter proceeds as follows: Section 4.1 formulates the centralized, multi-agent Markov decision process (MMDP) and derives the computational complexity of generating a solution. Section 4.2 describes an approximate approach to formulating MMDPs by decentralizing the problem and allowing each agent to carry a reduced-dimension model for each of its teammates rather than a full model for each. Section 4.3 similarly describes an alternate decentralized approximation where each agent lumps all of its teammates into a single model. Following these formulations, Section 4.4 describes the problem of choosing features when developing an approximate model and outlines several approaches for feature selection. We then discuss how to model the transition dynamics of these features in Section 4.5, and provide several approaches for doing so. In Section 4.6, we then compare our approximations with other approaches from the literature. Finally, in Section 4.7, we offer insights into how these algorithms will be implemented and what other factors should be considered when doing so.

4.1 Multi-agent Markov Decision Process (MMDP)

Multi-agent Markov decision processes (MMDPs)[70] are simply larger MDPs where the state and action spaces are often factored into smaller, local spaces per agent. Because of this formulation, MMDPs are an inherently centralized problem. The next few sections provide details on how an MMDP is formulated and how this formulation affects the size of the resulting problem.

4.1.1 MMDP State-Action Space, $(\mathcal{S} \times \mathcal{A})$

A state space is meant to fully represent the state of the agent, including all aspects and characteristics of the agent that are relevant to the underlying objective of the problem. In addition, MMDPs require that the state have the Markov property, which is that each state encapsulates all relevant information up to that point in time. This means that for the Markov property to hold, the next state can only be a function of the current state and action, nothing else. This has natural ties to the underlying cost function of the problem as it too may not be a function of past states or actions that are more than one timestep back.

When defining the state space of an MMDP, it is common practice to first generate a state vector that lists the attributes or characteristics of each agent that are relevant to the problem. For example, if we assume an agent's (x, y, z) location are relevant to the problem, the state vector $\mathbf{x}$ for a team of three agents would look like

$$\mathbf{x} = [x_1, y_1, z_1, x_2, y_2, z_2, x_3, y_3, z_3] \tag{4.1}$$

Then, after defining an alphabet of possible values for each element of the state vector, as

$$x_i \in X_i = \{0, 1, \ldots, 9\} \quad y_i \in Y_i = \{0, 1, \ldots, 19\} \quad z_i \in Z_i = \{0, 1, \ldots, 4\}$$

the full state space is then realized by taking all possible combinations of the values of the elements. This is known as the Cartesian product of the state vector and, for the example above with three agents, is written as

$$\mathcal{S} = X_1 \times Y_1 \times Z_1 \times X_2 \times Y_2 \times Z_2 \times X_3 \times Y_3 \times Z_3$$

which results in a state space of size $|\mathcal{S}| = (10 \cdot 20 \cdot 5)^3 = 1,000^3 = 1$ billion states.

Similarly, an agent's action space is meant to define all possible actions the agent is capable of taking. In a multi-agent setting, the action space typically becomes the Cartesian product of the agent's individual action spaces. Depending on the solution approach, one may wish to describe the state-action space jointly rather than individually. The remainder of this chapter adopts this convention as it will become clear that is enables easier comparison with the approximate methods developed later in this chapter. Hence, the complexity of the state-action space for an MMDP is: $\mathcal{O}(|\mathcal{S}||\mathcal{A}|)$ where the size of both $\mathcal{S}$ and $\mathcal{A}$ is exponential in the number of agents.

4.1.2 MMDP Transition Function, P

As in traditional MDPs, the transition function describes how the state is altered when specific actions are taken. In the general case, the state is altered stochastically, which means that taking a specific action from a specific state does not guarantee a single next-state but rather a probabilistic distribution over a set of possible next-states. It is this ability to describe and embed uncertainty in action outcome that makes MDPs such an attractive formulation for a variety of problems. Mathematically, the transition function P can be denoted in any number of way, such as $P(s'|s, a)$, but ultimately yields the probability of arriving in state s' given that action a was taken from state s. When represented in matrix form, P is a three-dimensional matrix of size $|\mathcal{S}| \cdot |\mathcal{A}| \cdot |\mathcal{S}|$, which for large problems is not feasible to represent in matrix form, but rather as a function whose inputs are the current state, current action and candidate next-state and whose output is the corresponding probability.

4.1.3 MMDP Cost Function, g

The cost function of the MDP is a fundamental element of the problem as it quantifies the qualitative "goodness" or "badness" of each state. When used in conjunction with the transition function P and discount factor α, one can calculate the "value" of each state using the Bellman equation

$$V(s) = \max_{a \in \mathcal{A}} \left(\sum_{s' \in \mathcal{S}} P(s'|s,a)(R(s,a) + \alpha V(s')) \right)$$

A state's "value", $V(s)$, is defined as the cost an agent can expect to incur over its lifetime when starting in that state and thereafter following the corresponding fixed-policy. Essentially what this means is that a state's "value" encapsulates the probabilistic "goodness" or "badness" of all the states that are reachable from the current state as well as a discounted version for those states that are reachable from the original reachable states and so on. This process is achieved in a number of ways, such as value-iteration or policy-iteration, both of which build the value function of the MDP.

4.2 Decentralized Multi-agent Markov Decision Process (Dec-MMDP)

In the multi-agent MDP setting described in the previous section, all agents are modeled identically within the MMDP. However, upon decentralizing the problem, each agent now has the choice of how to model its teammates. If a model identical to its own is used, the MMDP is recovered. The purpose of this section is to provide a method for approximating the model of an agent and to show how this model is central in the developement of a decentralized multi-agent Markov decision process (Dec-MMDP).

The approximation presented here is centered around independent agents that choose their own actions based on current knowledge of themselves and their teammates. Each Dec-MMDP is formulated from the perspective of a single agent, thus the full, n-agent problem is a collection of n Dec-MMDPs. However, as the term "decentralized" implies, each problem is solved independently such that each agent chooses its own actions. With this in mind, each Dec-MMDP is a tuple $\langle n, \mathcal{S}, \mathcal{A}, P, g, \alpha, \pi^{n=1} \rangle$ where n is the number of agents, $\mathcal{S}$ is the discrete state space of the problem, $\mathcal{A}$ is the local action space of the agent for which the Dec-MMDP is formulated, $P(s'|s,a)$ gives the transition probability from state s to state s' under action a, and $g(s,a)$ gives the cost of taking action a from state s. Future costs are discounted by a factor $0 < \alpha < 1$. Finally, $\pi^{n=1}$ is a fixed-policy that results from formulating and solving a single-agent MDP with identical local state, local action, transition dynamics and cost function.

Note that this formulation can be made more general by allowing for different state and action spaces and cost functions across the agents, however for the purposes of this thesis, we will assume that the agents are homogeneous in local state and action spaces as well as their local cost functions. Therefore, we can skip the subscripts on $\mathcal{A}$, P and g. A policy of the Dec-MMDP is denoted by $\pi : \mathcal{S} \to \mathcal{A}$, and maps each state of an agent's state space to a local action. Also, the action space of a Dec-MMDP is not joint, but rather contains only local actions.

Solving a Dec-MMDP is accomplished in an identical fashion as the centralized, MMDP case which seeks to minimize the problem's cost-to-go function, J_π, over the set of admissible policies Π, as shown here:

$$\min_{\pi \in \Pi} J_\pi(s_0) = \min_{\pi \in \Pi} \mathbb{E}\left[\sum_{k=0}^{\infty} \alpha^k g(s_k, \pi(s_k))\right].$$

In the following sections the elements of the Dec-MMDP tuple are formulated for agent i with the set Ω representing agent i's teammates (i.e. $\Omega \equiv \{1 \ldots n\} \setminus \{i\}$).

4.2.1 Dec-MMDP State-Action Space, $(\mathcal{S} \times \mathcal{A})$

In a factorized MMDP, the state-action space can be written as the Cartesian product of agent i's state-action space with that of its teammates, as shown by

$$\mathcal{S} \times \mathcal{A} = (\mathcal{S}_i \times \mathcal{A}_i) \times (\mathcal{S}_\Omega \times \mathcal{A}_\Omega). \tag{4.2}$$

Taking a careful look at Equation (4.2) above, we realize that the complexity of the state-action space is largely embedded within $(\mathcal{S}_\Omega \times \mathcal{A}_\Omega)$, which denotes the joint state-action space of all of agent i's teammates. This realization naturally leads to questions regarding the necessity of representing each teammate exactly and how sensitive the optimal solution is to approximations of this set. The decentralized multi-agent Markov decision process (Dec-MMDP) presented in this section aims to reduce this joint teammate state-action space to a lower-dimensional one using approximation.

While the Dec-MMDP formulation is decentralized, each agent still needs to carry a model of its teammates in order to promote the generation of cooperative plans and achieve low-cost solutions to the underlying optimization problem. Hence, there is an inherent cost/complexity tradeoff where a low-complexity teammate model naturally incurs a higher cost as it cannot produce the cooperation necessary for a low-cost solution. Conversely, a highly complex teammate model, e.g. one that matches the agent's own model, completely captures the inter-agent coupling can therefore generate cooperative behavior that leads to low-cost solutions. We seek the balance point where the model complexity is such that the problem can be solved in a reasonable amount of time while minimizing the cost-increase associated with a lower-dimensional teammate model.

To accomplish this balance, $\mathcal{S}_\Omega$ is the cartesian product of agent i's representation for each teammate j, as shown by $\mathcal{S}_\Omega = \prod_{j \in \Omega} \mathcal{S}_{ij}$. Therefore, due to the decentralized nature of the Dec-MMDP, $\mathcal{S}_{ij}$ denotes agent i's representation of teammate j's *approximated state-action* space, rather than just its state space alone. This is due to the fact that teammate actions are not explicitly represented in Dec-MMDPs, unlike MMDPs, and the complexity savings in this element are a direct result of their absence. The construction of $\mathcal{S}_{ij}$ is the mechanism by which the complexity of the resulting multi-agent planning problem can be reduced. The motivation behind this mechanism is the fact that in many multi-agent scenarios, each agent does not need to carry a high-fidelity model of its teammates. Rather, an abstract model can often yield minimal loss in terms of performance, while simultaneously accomplishing large gains in terms of reducing the computational requirements of generating a solution. In light of this, the designer of the Dec-MMDP formulation has the challenge of constructing $\mathcal{S}_{ij}$ in order to balance the loss in performance with gains in computational efficiency.

One approach to address this challenge is to use feature-based function approximation on the full teammate state-action space, $(\mathcal{S}_\Omega \times \mathcal{A}_\Omega)$, to remove state-action pairs that are unnecessary or unimportant for agent i to represent locally. Thus, $\mathcal{S}_{ij}$ would ideally be comprised of features that capture the reward/cost coupling between agent i and its teammates. The Dec-MMDP formulation uses the feature vector ϕ to attempt this.

$$\mathcal{S}_{ij} = \phi(s_j, \pi^{n=1}(s_j)), \tag{4.3}$$

where ϕ approximates agent j's state and its behavior as a vector of "features" and s_j is teammate j's local state (note that this implies full, individual observability). As the purpose of ϕ is to extract only the information relevant to the inter-agent coupling in the cost function, it is very similar to the features used in approximate dynamic programming [16] to quantize problems with large state spaces. This approach avoids enumerating the full cartesian product of agent i's and each of its teammates' full state spaces. Choosing a set of features is not a trivial task, and the performance of the approximate model depends on how well the approximation captures the full model. Section 4.4 is dedicated to this topic. The resulting complexity of the state-action space for an Dec-MMDP is: $\mathcal{O}\left(|\mathcal{S}_i||\mathcal{A}_i|\left(|\phi|^{n-1}\right)\right)$ where the size of ϕ depends on the number of features chosen to represent each teammate.

4.2.2 Dec-MMDP Transition Function, P

This section describes how action choice from a given state influences possible future states. It is important to note that this thesis assumes agents that are "transition independent". That is, the set of possible future states for agent i, given a start state and an action, is not influenced by the action choice of any other agent $j \in \Omega$. This assumption allows the transition function, P, to be written as

$$P = \begin{bmatrix} P_i & 0 \\ 0 & P_\Omega \end{bmatrix}, \tag{4.4}$$

where P_i transitions the local states and P_Ω transitions the features which represent the teammate states. Propagation of the local states is typically intuitive to the problem designer. However, much less intuitive are the transition dynamics of the features that represent the teammates. Also, it is unfortunately common that the problem solution is as sensitive to the accuracy of the feature transition model, P_Ω, as it is to the local transition model P_i. For this reason, Section 4.5 is dedicated to discussing this problem and presents several approaches for dealing with it.

4.2.3 Dec-MMDP Cost Function, g

The cost function of the Dec-MMDP formulation is conceptually identical to that of MDP and MMDP, namely a function of only the current state and action that providess a real-valued scalar indicating how "good" or "not good" the combination is. As the mission scenarios addressed in this thesis are primarily concerned with costs, this element is set up to penalize any undesirable outcomes in the formulation. However, the presence of approximations in forming $\mathcal{S}_\Omega$ remove some of the cost coupling between agents and cannot therefore "peak" into future possibilities the way the centralized problem can. The general cost function for multi-agent scenarios can typically be split into coupled and non-coupled costs as

$$g(s, a) = C_i + C_\Omega + C_G, \tag{4.5}$$

where C_i represents agent i's local costs based only on its local actions. On the other hand, C_Ω captures agent i's local costs that resulted from joint states and/or joint actions and C_G captures global costs affecting both agent i and its teammates.

4.3 Group-Aggregate Decentralized Multi-agent Markov Decision Process (GA-Dec-MMDP)

In this section, we present an extension to the Dec-MMDP formulation of Section 4.2 that enables the group of agent i's teammates to now be represented as an aggregate, rather than as individuals. Hence, we call this new approximation the group-aggregate Dec-MMDP (GA-Dec-MMDP). As the with the Dec-MMDP, each GA-Dec-MMDP is formulated from the perspective of a single agent, causing the full, n-agent problem to be a collection of n GA-Dec-MMDPs, each represented again by the tuple $\langle n, \mathcal{S}, \mathcal{A}, P, g, \gamma, \pi^{n=1} \rangle$. The following section discusses in the detail the formulation of the state-action space for the GA-Dec-MMDP as it differs from the Dec-MMDP formulated previously. However, the transition and cost functions remain identical to the Dec-MMDP formulation and are not duplicated here.

4.3.1 GA-Dec-MMDP State-Action Space, $(\mathcal{S} \times \mathcal{A})$

Like the Dec-MMDP, the GA-Dec-MMDP presented in this section also aims to reduce the joint teammate state-action space to a lower-dimensional one using function approximation. To do this, we again rely on a set of features, ϕ, to locally represent the large joint teammate state-action space. The difference here however, is that ϕ now approximates the group of teammates as a whole rather than a single teammate. Hence, the resulting complexity of the state-action space for a GA-Dec-MMDP is: $\mathcal{O}\left(|\mathcal{S}_i||\mathcal{A}_i||\phi|\right)$ where the size of ϕ depends on the number of features chosen to represent the set of all teammates.

4.4 Choosing Features

For the purposes of this research, a feature is essentially a basis function used in the approximation of another, more complex function. Specificcally in the context of the persistent health missions that are the topic of this thesis, a feature is an abstraction or conglomerate of possibly multiple states. The purpose of such a feature is to provide a more compact representation of an overarching function or space. Choosing a set of features to approximate a larger set of states is a non-trivial task, as the set of features with which the states could be described is typically large. Feature selection is the process of determining which subset of the possible features should be included so as to result in the best performance and lowest computational complexity. The challenge is that the number of potential subsets grows exponentially with respect to the number of features [60]. Hence, including unnecessary features results in increased computation while excluding important features will clearly reduce the quality of the resulting solution [83]. For these reasons, feature selection remains an important and very active area of research in the machine learning and artificial intelligence communities [33, 57, 60, 80, 83, 111].

As the task of automatic feature selection is indeed so challenging, it is often accomplished via an experienced problem designer with vast and insightful domain knowledge. However, even without such knowledge there are a few resources available that can act as guides in determining the *right* set of features for the desired approximation, including the (GA)-Dec-MMDP approximations given above. For instance, the value-function, or Q-function, associated with the problem of interest provides much insight into the relationship of each state with those that are reachable from it. Unfortunately, having the value- or Q-function means that the problem has already been solved. We are, however, still formulating the problem, and therefore have no such value- or Q-function available. So, we could instead analyze the value-function produced by a similar problem formulation. Or, though admittedly less insightful than the value-function, we could use the underlying cost function of the problem to guide our feature selection process.

One of the main benefits of using the cost function as a guide, see for example Equation (4.5), is that the designer can immediately gain insight into the sensitivity of $g(s, a)$ with respect to each of the components of C_Ω. Armed with this knowledge, the problem designer can then build the feature set ϕ such that the state-action pairs within $(\mathcal{S}_\Omega \times \mathcal{A}_\Omega)$ that most influence $g(s, a)$ are captured, while those that have little or no impact are discarded.

4.5 Propagating Features

Once a set of features has been selected, the remaining challenge is to express their transition dynamics. This can be tricky as features often have a less-intuitive meaning than states or actions. Additionally, general methods for modeling the transition dynamics of features is a topic that is not well-covered in the literature. We present three candidate approaches for generating the feature transition dynamics, given in Equation (4.4) as P_Ω. In the first approach, we solve the original, un-approximated problem of interest for the case of fewer agents, thereby making the computational complexity just small enough that policy construction becomes feasible on a reasonable time-scale. We then run virtual trajectories through this policy and build a history of virtual states. For each snapshot of the virtual state in the history, we can generate the appropriate feature(s), thus creating a history of virtual features. A data-driven model for the feature transitions can then be created from this history of features. This model must then be somehow *scaled* up to fit problem formulations with more agents. Details regarding the implementation of this method can be found in Section 5.3.3.

A second approach for generating the feature transition dynamics is similar to the first, but uses a policy generated through a heuristic planner for the proper number of agents, so post-process scaling is not required.

Finally, the third approach is an iterative process to generate the feature transition dynamics. After an initial *guess* at how the feature transitions should look (this guess can be generated using either of the first two approaches) the iteration consists of the following steps:

1. Use the current feature-transition model to solve the problem of interest, thereby constructing a policy

2. Similar to the first two cases, we then run virtual state trajectories through this policy to create a history of virtual states, from which a candidate feature-transition model is built directly

3. This candidate model then becomes the current feature-transition model

If the problem parameters are known and static, any of these approaches only needs to be performed once, offline, prior to solving the approximate problem of interest. However, when the parameters are dynamic and unknown, the construction of P_Ω needs to be brought into the inner-loop and thereby be solved/re-solved online. In which case the computational complexity of generating a solution grows accordingly.

4.6 Alternative Approaches

As stated in Section 2.2.5, the general problem we are interested in solveing is the decentralized Markov decision process (Dec-MDP). There are other approaches for approximating the full Dec-MDP other than the MMDP and the two approximations outlined above. Most notable among these are the decentralized transition-independent Markov decision process (TI-Dec-MDP) [9, 10] and the sparse-interaction decentralized Markov decision process (SI-Dec-MDP) [70, 71].

TI-Dec-MDPs are introduced in [10] and assume, as the name suggests, transition indepence of each of the participating agents. Transition independence is defined as when an agent's dynamics are unaffected by any other agent's state-action pair. That is, agent i is transition independent if:

$$P(s_i'|s_i, a_i, s_\Omega, a_\Omega, s_\Omega') = P(s_i'|s_i, a_i)\ \forall i \in \{1 \dots n\}$$

Central to this approach is the notion of an *event*, which is a desired snapshot of the system state and possibly also the joint action. After creating this list of possible events, along with the probability of each one occurring and its associated reward/-cost, then multiple TI-Dec-MDPs can be instantiated and handed to a coverage-set algorithm that decides which instances to include in the problem formulation. In [112], a reward dependent TI-Dec-MDP is reformulated into a maximization problem with non-linear constraints so as to avoid the exponential increase in complexity. The constraints are then discretized into piecewise linear chunks and a mixed-integer linear program with hill-climbing search is formed which can be solved using any off-the-shelf solver. In [81], a TI-Dec-MDP is reformulated as a separable bilinear program and issues with the coverage set algorithm (CSA), which is needed in the solution of TI-Dec-MDPs, are brought forth. Namely that there are no error bounds associated with CSA and, although it typically gives good anytime performance, it is shown to be numerically unstable and have exponential complexity in the number of best-response policies. So, the authors reformulate the CSA algorithm and provide an online error bound for the resulting approximate solutions. Finally, in our own experiences, the required set of *events* can be very cumbersome to develop. And, although the computational complexity is linear in the size of the event list, this list can easily reach the order of magnitude of the state space to accurately describe the problem of interest.

SI-Dec-MDPs [71] are a subclass of Dec-MDPs that attempt to factor the joint state space into two disjoint sets: states where agents are independent and states where they can interact. In regions where an agent is independent, a single-agent MDP is used for its decision making. In the interaction areas, full observability of all interacting agents allows for an MMDP to be formed around these joint interaction states only, and it is then used for local decision making [70, 71]. In [100], the decentralized multi-agent planning problem is cast as a Markov game, but is similar to SI-Dec-MDP as agent interactions are modeled as local and a differentiation is made between where agents interact and where they act independently - which reduces the complexity from NEXP-Complete to NP-Complete. The key differences between SI-Dec-MDP and the (GA-)Dec-MMDP developed above are how the state space is divided and what information is shared at any given step.

4.7 Implementation Considerations

As mentioned in Section 4.2, a collection of n (GA-)Dec-MMDPs are needed to model and solve a n-agent cooperative planning problem. As the (GA-)Dec-MMDP approximations assume full, individual observability, the agents must be able to communicate when implemented in practice. This communication is in two forms: local state and intended action. As each agent shares its local state, all agents can re-construct the full system state locally. Each agent then translates the portion of this state regarding its teammates into corresponding features. Each agent then chooses its own action based on its own state and an internal prediction of what each of its teammates action choice will be (or, what the aggregate teammate action will be in the case of GA-Dec-MMDP). A consensus loop must then occur until the predicted teammate actions match those broadcast by the teammates themselves.

Chapter 5

Persistent Surveillance Mission

This chapter focuses on detailing an extension of the persistent surveillance mission (PSM) first proposed and developed by Bethke et al. [21] as the mission scenario under which the architecture of Chapter 3 and the cooperative planning algorithms of Chapter 4 are to be implemented. The mission involves a group of N autonomous agents, each equipped with some type of sensor and initially situated at some base location, as shown in Figure 5-1. Their objectives are to continuously survey a pre-specified region of interest and to closely track any objects of interest discovered there. The problem becomes challenging when stochastic models for sensor and actuator health are included in the formulation as well as stochastic dynamics for fuel consumption.

As seen in Figure 5-1, the mission area is divided into three distinct regions geographicallty. These regions are labeled as the *Base*, *Communication* and *Task* areas. Aerial agents (UAVs) start at the base area and travel from there to other regions for tasking and communication duties. As fuel depletes, or failures occur, these agents must return to base for refueling and/or repair. The communication area is a transition region between the base and tasking areas and is where an agent can act as a relay link for communication between the base and tasking areas. This communication area also serves as a base for a team of ground agents (UGVs). In the tasking area, several target vehicles are hidden among a number of neutral vehicles - this is where the persistent surveillance and tracking takes place. The objective of the mission is to search for target vehicles in the tasking area while continuously tracking

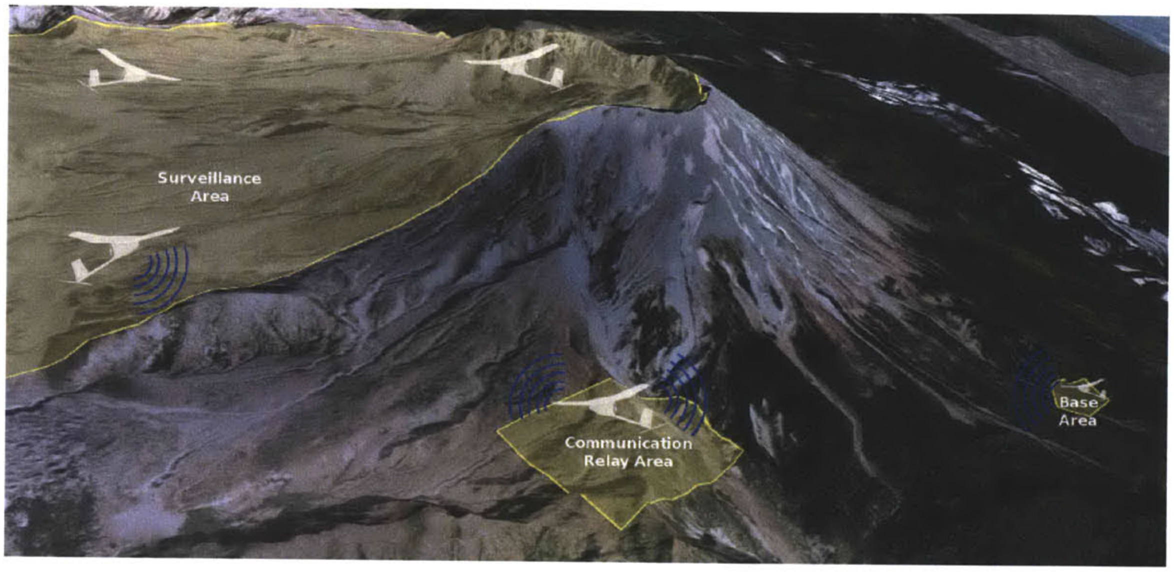


Figure 5-1: Mission scenario: N autonomous agents cooperate to continuously survey a specified region and to track any objects of interest discovered there while maintaining constant communication with the base location. This behavior is to be persistently maintained even under sensor, actuator and battery health degradations.

those that have already been detected. Targets can be discovered and tracked by both UAVs and UGVs. However, news of the discovery can only reach the agents at the base station if an additional UAV is located in the communication area to act as a relay.

Vehicle dynamics provide a number of interesting health management aspects to the problem. In particular, management of fuel is an important concern in extended-duration missions such as the persistent surveillance problem. The UAVs have a specified maximum fuel capacity F_{max}, and we assume that the rate at which they burn fuel, $\dot{F}_{burn}$, may vary randomly during the mission due to bursts of aggressive maneuvering, engine wear and tear, adverse environmental conditions, damage sustained during flight, etc. Thus, the total flight time each vehicle achieves on F_{max} units of fuel is a random variable, and this uncertainty must be accounted for in the problem formulation. If a vehicle runs out of fuel while in flight, it is considered *Crashed* and is no longer available for tasking. The vehicles can refuel however, at a rate $\dot{F}_{refuel}$ by returning to the base location where a battery changing/charging station automates the refueling process.

Moreover, each agent has a non-zero probability of experiencing a sensor failure during the mission. Such a failure may decrease an agent's capability to below that which is required for successful completion of its task(s). For instance, a vehicle with a failed sensor cannot effectively search for, or track, any of the targets in the tasking area. However, it can still act as a communication relay. Similarly, there exists a non-zero probability that an agent may experience an actuator degradation that decreases its maneuverability such that it can no longer be used for target-tracking, but only for searching or as a communication relay. However, unlike crashes, all failures and degradations can be repaired once the vehicle returns to the base location.

Based on the problem description above, plans could be explicitly described to induce and encourage inter-agent cooperation. However, heuristic approaches such as this are not robust to events such as sensor failures, actuator degradations and uncertain fuel use. Therefore, it is desirable to model the mission as a stochastic optimal control problem which can be initially solved off-line to extract a reasonable policy without explicitly forcing a heuristic strategy. Furthermore, under the iCCA architecture, online adaptation of planner parameters becomes possible and, when combined with online-solvable planning algorithms such as the Dec-MMDPs and GA-Dec-MMDPS presented in the previous chapter, results in an adaptive planning system that learns to improve over time and with experience.

5.1 PSM Formulated as an MMDP

An infinite-horizon, discounted multi-agent MDP (MMDP) is specified by $\langle n, \mathcal{S}, \mathcal{A}, P, g, \alpha \rangle$, where n is the number of agents, $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $P(s'|s,a)$ gives the transition probability from state s to state s' under action a, and $g(s,a)$ gives the cost of taking action a in state s. We assume that the MDP model is known. Future costs are discounted by a factor $0 < \alpha < 1$. A policy of the MDP is denoted by $\pi : \mathcal{S} \rightarrow \mathcal{A}$, and is a mapping of states to actions. Given the MDP specification, the problem is to minimize the cost-to-go function J_π over the set of admissible policies Π, as shown here:

$$\min_{\pi \in \Pi} J_\pi(s_0) = \min_{\pi \in \Pi} \mathbb{E}\left[\sum_{k=0}^{\infty} \alpha^k g(s_k, \pi(s_k))\right].$$

For notational convenience, the cost and state transition functions for a fixed policy π are defined as $g_s^\pi \equiv g(s, \pi(s))$, $P_{ss'}^\pi \equiv P_{ss'}(\pi(s))$ respectively. The cost-to-go for a fixed policy π satisfies the Bellman equation [16]

$$J_\pi(s) = g_s^\pi + \alpha \sum_{s' \in \mathcal{S}} P_{ss'}^\pi J_\pi(s') \quad \forall s \in \mathcal{S}, \tag{5.1}$$

which can also be expressed compactly as $J_\pi = T_\pi J_\pi$, where T_π is the (fixed-policy) dynamic programming operator.

5.1.1 State Space $\mathcal{S}$

The state of each UAV is given by three scalar variables describing the vehicle's location, fuel remaining and health status. The location of agent i is denoted as y_i, where

$$y_i \in Y = \{Y_B, Y_C, Y_S\} \tag{5.2}$$

where Y_B is the *Base* area, Y_C is the *Communication Relay* area, and Y_S is the *Surveillance* area shown in Figure 5-1. Similarly, the fuel state of agent i, denoted f_i, is described by a discrete set of possible fuel quantities,

$$f_i \in F = \{0, \Delta f, 2\Delta f, \ldots, F_{max} - \Delta f, F_{max}\} \tag{5.3}$$

where Δf is an appropriate discrete fuel quantity ($\Delta f = 1$, and $F_{max} = 10$ for this research). Agent i's health status, h_i, is described by a discrete set of possible health states, given by

$$h_i \in H = \{H_{nom},\ H_{sns},\ H_{act}\}. \tag{5.4}$$

where H_{nom}, H_{sns} and H_{act} respectively represent nominal health, a failed sensor and a damaged actuator.

The total system state vector $\mathbf{x}$ for the centralized formulation is thus given by cross product of the states y_i, f_i and h_i (Equations (5.2)-(5.4)) for each UAV, as shown by $\mathbf{x} = [y_i\ f_i\ h_i,\ \ldots,\ y_n\ f_n, h_n]$. The size of the state space is found by counting all possible realizations of $\mathbf{x}$, which yields $|\mathcal{S}| = (|Y| \times |F| \times |H|)^n$.

5.1.2 Control Space $\mathcal{A}$

The actions available to each agent in general are $u \in \{-1,\ 0,\ +1\}$, which correspond to {"Move toward *Base*", "Stay", "Move toward *Surveillance*"} respectively. However, the specific controls u_i available for the i^{th} agent depend on the agent's current location y_i and its remaining fuel f_i, according to the following rules:

$$u_i \in \begin{cases} \{-1,\ 0,\ +1\}, & \text{if } y_i = Y_C \\ \{-1,\ 0\}, & \text{if } y_i = Y_S \\ \{0,\ +1\}, & \text{if } y_i = Y_B \\ \{0\}, & \text{if } f_i = 0 \end{cases} \tag{5.5}$$

The total system action vector $\mathbf{u}$ for the centralized formulation is given by the cross product of u_i for each agent, as shown by $\mathbf{u} = [u_i,\ \ldots,\ u_n]$. The size of the action space is found by counting all possible realizations of $\mathbf{u}$, which yields $|\mathcal{A}| = (|u_i|)^n = 3^n$.

5.1.3 State Transition Model P

The state transition model P captures the qualitative description of the dynamics given at the start of this section. The model for agent location y_i can be factored into dynamics for each individual agent, which are deterministic and are described by the following rules:

$$y_i(k+1) = \begin{cases} y_i(k), & \text{if } f_i = 0 \text{ or } u_i(k) = 0 \\ Y_B, & \text{if } y_i(k) = Y_C \text{ and } u_i(k) = -1 \\ Y_S, & \text{if } y_i(k) = Y_C \text{ and } u_i(k) = +1 \\ Y_C, & \text{if } y_i(k) = Y_S \text{ and } u_i(k) = -1 \\ Y_C, & \text{if } y_i(k) = Y_B \text{ and } u_i(k) = +1 \end{cases} \tag{5.6}$$

The dynamics for the fuel state f_i are stochastic with parameter p_{fuel} representing the probability of burning fuel at the nominal rate $\dot{F}_{burn}$. So, with probability p_{fuel}, fuel is consumed at a rate of $\dot{F}_{burn}$ per time step. And with probability $1 - p_{fuel}$, fuel is consumed at twice the nominal rate. Specifically, this model evolves according to the following rules:

$$f_i(k+1) = \begin{cases} 0, & \text{if } f_i(k) = 0 \\ f_i(k) + \dot{F}_{refuel}, & \text{if } y_i(k) = Y_B \text{ and } u_i(k) \in \{-1,\ 0\} \\ f_i(k), & \text{if } y_i(k) = Y_B \text{ and } f_i(k) = F_{max} \\ f_i(k) - \dot{F}_{burn}, & \text{if } y_i(k) \neq Y_B \quad Prob = p_{fuel} \\ f_i(k) - 2\dot{F}_{burn}, & \text{if } y_i(k) \neq Y_B \quad Prob = 1 - p_{fuel} \end{cases} \tag{5.7}$$

The health state of each agent is also a stochastic model with parameters p_{sns} and p_{act} representing the probability of a sensor failure, and actuator damage respectively.

This health model evolves according to the following rules:

$$h_i(k+1) = \begin{cases} h_i(k), & \text{if } f_i = 0 \\ H_{nom}, & \text{if } y_i(k) = Y_B \\ H_{nom}, & \text{if } y_i(k) \neq Y_B \text{ and } h_i(k) = H_{nom} \quad Prob = (1 - P_{sns})(1 - P_{act}) \\ H_{sns}, & \text{if } y_i(k) \neq Y_B \text{ and } h_i(k) = H_{nom} \quad Prob = P_{sns}(1 - P_{act}) \\ H_{act}, & \text{if } y_i(k) \neq Y_B \text{ and } h_i(k) = H_{nom} \quad Prob = P_{act} \end{cases} \tag{5.8}$$

5.1.4 Cost Function g

The cost function $g(\mathbf{x}, \mathbf{u})$ is set up to penalize any undesirable outcomes in the mission, and is characterized by Equation (5.9). For the persistent surveillance mission, the undesirable outcomes include: (1) Having less than the desired number of agents in the surveillance region, and (2) Having no communication relay. The first undesirable outcome results in a small cost of C_{gap} for each of the δ_S missing agents, where $\delta_S = \max((n_d - n_S), 0)$, n_d is the desired number of agents, and n_S is the actual number of healthy agents in the surveillance area. Note that we assume the existence of an indicator function $\text{comm}(\mathbf{x})$, which returns 1 if a communication relay is present and 0 otherwise. Finally, failure to provide a communication relay *or* to maintain any agents in the surveillance area results in a high cost of C_{fail}. Combined, the cost function can be expressed as:

$$g(\mathbf{x}, \mathbf{u}) = \begin{cases} 0 & \text{if comm}(\mathbf{x}) = 1 \text{ and } n_S = n_d \\ C_{gap} \cdot \delta_S & \text{if comm}(\mathbf{x}) = 1 \text{ and } n_S > 0 \text{ and } n_S < n_d \\ C_{fail} & \text{otherwise} \end{cases} \tag{5.9}$$

As seen, the cost is zero only when the desired number of healthy agents are in the surveillance area *and* a communication relay is provided.

5.2 PSM Formulated as a Dec-MMDP

The MMDP formulated in the previous section, inherently captures all inter-agent coupling while exhaustively searching the joint state-action space for the specific state-action combinations that result in optimal planner output. However, this approach is known to not scale well in the number of agents and, in practice, is slow to solve even for the case of the persistent search and track mission with 3 agents.

Motivated by the need to construct and adapt planner output in real-time (or approximate such output as accurately as possible), it is worthwhile to consider alternate problem formulations to the full, centralized case, such as a decentralized formulation [14, 15, 70, 95]. In particular, decentralized planners that scale well while yielding results that are close to optimal are particularly appealing. Additionally, staying near stochastic optimal control is also appealing due to the natural integration of uncertainty within the problem formulation.

As a result, one promising field of research that addresses these issues is approximate dynamic programming. Essentially, the goal of any approximate dynamic programming method applied to a multi-agent planning problem is to capture the inter-agent coupling with a minimal number of states. In many scenarios, including the persistent search and track mission here, inter-agent coupling is limited to the cost function (see Section 5.1.4) as state-transition dynamics are completely decoupled (see Section 5.1.3).

How well this inter-agent cost-coupling is captured has a large effect on the optimality of the solution. Unfortunately, capturing this coupling also causes an increase in problem size and therefore computational complexity. What is largely missing is a knob that gives the problem designer control over the inherent trade-off between problem size (e.g. solution speed) and the level of inter-agent coupling captured in the formulation (e.g. solution optimality). Motivated by this, we formulate a decentralized multi-agent Markov decision process (Dec-MMDP), the solution of which suggests an action for a single agent based on its local state and a feature-based abstraction/approximation of its teammate's local states. The Dec-MMDP provides the missing control knob in the form of how agent i models its teammates.

A Dec-MMDP is a tuple $\langle n, \mathcal{S}, \mathcal{A}, P, g, \alpha, \pi^{n=1} \rangle$ where n is the number of agents, $\mathcal{S}$ and $\mathcal{A}$ are again the state and action spaces respectively but both will be constructed differently from the MMDP formulation. As before, $P_{ij}(u)$ gives the transition probability from state i to state j under action u, and $g(i, u)$ gives the cost of taking action u in state i. Future costs are still discounted by a factor $0 < \alpha < 1$. Finally, $\pi^{n=1}$ is a fixed-policy that results from a single-agent MDP (formulated identically as in Section 5.1 with $n = 1$). It should be noted here that it takes a collection of n Dec-MMDPs to solve the same problem as a single MMDP. This is due to the fact that each Dec-MMDP is formulated for a single agent, as is evident in the form of the action space. However, each Dec-MMDP is solved independently.

This formulation can be made more general by allowing for different action spaces and cost functions across the n Dec-MMDP instances. However, for the purposes of this thesis, the agents are homogeneous, which enables the convenient dropping of the subscripts on $\mathcal{A}$, P and g. A policy of the Dec-MMDP is denoted by $\pi : \mathcal{S} \rightarrow \mathcal{A}$, and is a mapping of states to actions. In this case, however, the state is not necessarily full and the actions are not joint, but rather single-agent actions. The problem is still solved identically to the centralized case, seeking to minimize the cost-to-go function J_π over the set of admissible policies Π, as shown here:

$$\min_{\pi \in \Pi} J_\pi(i_0) = \min_{\pi \in \Pi} \mathbb{E}\left[\sum_{k=0}^{\infty} \alpha^k g(i_k, \pi(i_k))\right].$$

The next sections detail the components of the Dec-MMDP as formulated for agent i, with its set of teammates denoted as Ω, where $\Omega \equiv \{1 \ldots n\} \setminus \{i\}$.

5.2.1 State Space $\mathcal{S}$

When transitioning to the decentralized case, the state space of agent i must somehow account for the intended actions of its teammates as the formulation no longer enumerates the joint action set. To accommodate this, agent i's state space is factorized in the following manner:

$$\mathcal{S} = \mathcal{S}_i \times \mathcal{S}_\Omega, \tag{5.10}$$

where $\mathcal{S}_i$ is agent i's local state space, such that

$$\mathcal{S}_i = Y \times F \times H. \tag{5.11}$$

The joint state space of agent i's teammates is given by $\mathcal{S}_\Omega$, where

$$\mathcal{S}_\Omega = \prod \mathcal{S}_{ij},\ \forall j \in \Omega, \tag{5.12}$$

where $\mathcal{S}_{ij}$ is to be understood as agent i's representation of agent j's local state-action space. Obviously, if $S_{ij} = (Y \times F \times H \times \mathcal{A})$, this would result in a problem that is computationally intractable even for the case of $n = 3$, similar to the MMDP formulation for that case. To avoid this issue of dimensionality, we implement a state aggregation approach to construct S_{ij} in a way that approximates (or, aggregates) the state-action space of agent i's teammates.

Hence, the construction of $\mathcal{S}_{ij}$ is the knob that gives the problem designer control over the trade-off between problem size and the level of inter-agent coupling captured in the formulation. The fully-coupled problem can be re-captured by letting $\mathcal{S}_{ij} = \mathcal{S}_j \times \mathcal{A}$, where $S_j = Y \times F \times H$, allowing agent i a full description of every other agent, including their intended actions. Alternatively, agent i can completely ignore his teammates, and thus formulate a single-agent MDP, by letting $\mathcal{S}_{ij} = \emptyset$. Thus, $\mathcal{S}_{ij}$ represents a mechanism for approximating the state-action space of each of agent i's teammates, whereby solution optimality can potentially be sacrificed for a reduction of the size of state space ($|\mathcal{S}|$). One approach for constructing $\mathcal{S}_{ij}$ is by using linear approximation of the form

$$\mathcal{S}_{ij} = \phi(s_j, \pi^{n=1}(s_j)), \tag{5.13}$$

where ϕ approximates agent j's state and its behavior as a vector of "features". The purpose of ϕ is to extract only the information relevant to the inter-agent coupling in the cost function, yielding a set of features. In this sense, ϕ is very similar to the features used in approximate dynamic programming[16] to quantize problems with large state spaces. The $\pi^{n=1}$ function is a fixed policy used to predict the intended action of each teammate. In this example, the following set of mutually exclusive, binary features are used:

$$\phi = \{Y_B^+,\ Y_C^-,\ Y_C^0,\ Y_C^+,\ Y_S^-,\ Y_S^0\},\ \text{where} \tag{5.14}$$

- Y_B^+ denotes teammate j as being in the *Base* location, moving toward *Surveillance*,
- Y_C^- means j is in the *Communication* location, moving toward *Base*,
- Y_C^0 means j is in the *Communication* location, staying put,
- Y_C^+ means j is in the *Communication* location, moving toward *Surveillance*,
- Y_S^- means j is in the *Surveillance* location, moving toward *Base*, and
- Y_S^0 denotes j as being in the *Surveillance* location, staying put.

This set of features essentially constitutes a one-step approximation to the full, infinite-horizon problem. The total system state vector $\mathbf{x}$ for the decentralized formulation is thus given by outer product of the state vector $\mathbf{x} = [y_i\ f_i\ h_i,\ \mathcal{S}_\Omega]$. The size of the state space is found by counting all possible realizations of $\mathbf{x}$, yielding $|\mathcal{S}| = (|Y| \times |F| \times |H|) \times |\mathcal{S}_{ij}|^{n-1}$.

5.2.2 Control Space $\mathcal{A}$

The control space in the decentralized problem differs from the centralized formulation in that here it is for a single agent. However, even if the control space were joint, since the construction of the state space allows for approximating teammate states, it cannot be guaranteed that the actions agent i chooses for his teammates will be the same actions they will choose for themselves. Otherwise, the actions available to agent i are identical to those formulated in the centralized Section 5.1.2 above. The total system action vector $\mathbf{u}$ for the decentralized formulation is simply $\mathbf{u} = u_i$, leaving the size of the action space $|\mathcal{A}| = (|u_i|)$.

5.2.3 State Transition Model P

The state transition model for $\mathcal{S}_i$ is identical to that in 5.1.3. For $\mathcal{S}_j$, however, transitions according to the following model:

$$s_{ij}(k+1) = \begin{cases} Y_C^0, & \text{if } s_j(k) = Y_B^+ \quad Prob = 0.5 \\ Y_C^+, & \text{if } s_j(k) = Y_B^+ \quad Prob = 0.5 \\ Y_B^+, & \text{if } s_j(k) = Y_C^- \\ Y_C^0, & \text{if } s_j(k) = Y_C^0 \\ Y_S^0, & \text{if } s_j(k) = Y_C^+ \\ Y_C^-, & \text{if } s_j(k) = Y_S^- \quad Prob = 0.5 \\ Y_C^0, & \text{if } s_j(k) = Y_S^- \quad Prob = 0.5 \\ Y_S^0, & \text{if } s_j(k) = Y_S^0 \end{cases} \tag{5.15}$$

5.2.4 Cost Function g

The cost function $g(\mathbf{x}, \mathbf{u})$ in the decentralized case is also set up to penalize any undesirable outcomes in the mission, where the undesirable outcomes still include: (1) Having less than the desired number of agents in the surveillance region, and (2) Having no communication relay. Combined, the cost function is again expressed as:

$$g(\mathbf{x}, \mathbf{u}) = \begin{cases} 0 & \text{if comm}(\mathbf{x}) = 1 \text{ and } n_S = n_d \\ C_{gap} \cdot \delta_S & \text{if comm}(\mathbf{x}) = 1 \text{ and } n_S > 0 \text{ and } n_S < n_d \\ C_{fail} & \text{otherwise} \end{cases} \tag{5.16}$$

However, while identical to the centralized case, the presence of approximations through $\mathcal{S}_{ij}$ remove some of the cost coupling between agents and the cost function cannot therefore "peak" into future possibilities the way the centralized problem can. This is due to the fact that agent i can only use elements of its represented state $\mathcal{S} = \mathcal{S}_i \times \mathcal{S}_\Omega$ and local actions to determine cost. Remembering that $\mathcal{S}_\Omega$ approximates the state-action space of all its teammates down to an $n-1$ element vector ϕ, agent i can only "guess" when calculating n_S and determining the output of the comm($\mathbf{x}$) indicator function. Made according to the rules in Equation (5.17), these "guesses" result in sub-optimal behavior, and therefore incur more cost than in the centralized case.

$$\begin{aligned} n_S &= n_S + 1, \quad \text{if } s_j = Y_S^0 \\ \text{comm}(\mathbf{x}) &= 1, \quad \text{if } s_j = Y_C^\star \\ \text{comm}(\mathbf{x}) &= 0, \quad \text{otherwise} \end{aligned} \tag{5.17}$$

5.3 PSM Formulated as a GA-Dec-MMDP

Motivated by the need to construct and adapt planner output in as close to real-time as possible, previous work has considered alternate problem formulations where approximations were introduced in the formulation itself, rather than applied to the solution approach, e.g. decentralized sparse-interaction MDPs [69] and decentralized multi-agent MDPs (Dec-MMDPs) [87].

In a previous decentralized approximation (see [87]), each agent represented its teammates with a reduced-dimensional model, generated using state aggregation on the full model. The focus of this section is to extend this approach and allow each agent to approximate all of its teammates collectively with a single, reduced model. This model is generated using aggregation techniques on the joint state-action space of the teammates. The motivation for this approach comes from the decision-making perspective of an individual agent: that as long as someone satisfies mission goals/-constraints, it does not need to know specifically who, or how. Thus, the aggregated state of all of an agent's teammates is reduced to a combination of features, such as the total number of agents in surveillance area, or whether another agent will act as the communication relay. The particular advantage of this formulation is that the growth of the size of the state space can be made linear in the number of agents, rather than exponential.

A GA-Dec-MMDP is a tuple $\langle n, \mathcal{S}, \mathcal{A}, P, g, \alpha, \pi^{n=1} \rangle$ where n is the number of agents, $\mathcal{S}$ and $\mathcal{A}$ are again the state and action spaces and $P_{ij}(u)$ again gives the transition probability from state i to state j under action u. As before, $g(i,u)$ gives the cost of taking action u in state i and future costs are still discounted by a factor $0 < \alpha < 1$ and $\pi^{n=1}$ remains a fixed policy that results from a single-agent MDP. The differences are in how $\mathcal{S}$, $\mathcal{A}$ and P are constructed, which is outlined in the following sections where the components of the GA-Dec-MMDP are formulated for agent i, with its set of teammates denoted as $\Omega \equiv \{1 \dots n\} \setminus \{i\}$.

5.3.1 State Space $\mathcal{S}$

In the GA-Dec-MMDP formulation, the state space is factored as:

$$\mathcal{S} = \mathcal{S}_i \times \mathcal{S}_\Omega$$

where $\mathcal{S}_i$ denotes the local state of agent i and $\mathcal{S}_\Omega$ represents the collective state-action space for all of agent i's teammates, or the *group-aggregate* state.

For the PSM mission, the local state of each agent is given by three scalar variables describing the agent's location, fuel remaining and health status. The location of agent i is denoted as y_i, where

$$y_i \in \{Y_B, Y_C, Y_S\} \tag{5.18}$$

where Y_B is the *Base* area, Y_C is the *Communication Relay* area, and Y_S is the *Surveillance* area shown in Figure 5-1. Similarly, the fuel state of agent i, f_i, is described by a discrete set of possible fuel quantities,

$$f_i \in \{0, \Delta f, 2\Delta f, \dots, F_{max} - \Delta f, F_{max}\} \tag{5.19}$$

where Δf is an appropriate discrete fuel quantity. Agent i's health status, h_i, is described by a discrete set of possible health states, given by

$$h_i \in \{H_{nom},\ H_{sns},\ H_{act}\} \tag{5.20}$$

where H_{nom}, H_{sns} and H_{act} respectively represent nominal health, a failed sensor and a damaged actuator. Combining these parts, an agent's local state space, $\mathcal{S}_i$, is defined by the cross product of the states y_i, f_i and h_i, which yields

$$\mathcal{S}_i = [y_i \times f_i \times h_i]$$

The purpose of the group aggregate state, $\mathcal{S}_\Omega$, is to compactly represent all of agent i's teammates. Although the content of $\mathcal{S}_\Omega$ is problem-dependent, the objective function provides a guideline for its construction. For example, in the PSM mission, the objective is to avoid gaps in the coverage of the surveillance and communication regions. To accomplish this, the agents must coordinate their actions, keeping their own health and the health of their teammates in mind. The optimal way to achieve this coordination in the presence of uncertainties, is by formulating the problem as an MMDP and solving it exactly. However, this is intractable for teams larger than $n = 3$, for the PSM mission. So, to avoid enumerating all possible combinations of y_i, f_i and h_i for each teammate, agent i aggregates the state-action spaces of its teammates into a set of features using ϕ

$$\mathcal{S}_\Omega = \phi(\mathcal{S}_{ij}),\ \forall j \in \Omega, \tag{5.21}$$

where $\mathcal{S}_{ij}$ is agent i's representation of agent j's local state-action space and ϕ is a function that extracts information relevant to the inter-agent coupling in the cost function. Forward propagation of the state of each teammate is accomplished using agent i's perception of their local state and agent i's non-cooperative policy $\pi^{n=1}$. In other words, agent i predicts each teammate's action based on what he would do if his teammate's local state were his own. Agent i then evaluates these predicted teammate states against elements of his local cost function to construct $\mathcal{S}_\Omega$. Hence, the aggregate state is written as

$$\mathcal{S}_\Omega = [\mathbf{I}_{[n_c>0]} \times n_s] = \begin{bmatrix} 0 & 0 \\ \vdots & \vdots \\ 0 & n-1 \\ 1 & 0 \\ \vdots & \vdots \\ 1 & n-2 \end{bmatrix} \tag{5.22}$$

where n_c denotes the number of teammates in the communication area and $n_s \in \{0,\ n-1\}$ the number of teammates in the surveillance area. $\mathbf{I}_{[n_c>0]} \in \{0,\ 1\}$ is an indicator function on whether the communication requirement is satisfied. The system state vector $\mathbf{x}$ for this formulation is given by $\mathbf{x} = [y_i\ f_i\ h_i,\ \mathcal{S}_\Omega]$. The size of the state space is found by counting all possible realizations of the state vector $\mathbf{x}$, yielding $|\mathcal{S}| = (|Y| \times |F| \times |H|) \times |\mathcal{S}_\Omega|$, which scales linearly in n as $|Y|$, $|F|$, and $|H|$ are constants and $|\mathcal{S}_\Omega| = 2n$.

5.3.2 Action Space $\mathcal{A}$

The control space differs from the centralized formulation in that here it is for a single agent. However, even if the control space were joint, since the construction of the state space allows for approximating teammate states, it cannot be guaranteed that the actions agent i chooses for his teammates will be the same actions they will choose for themselves. Otherwise, the actions available to each agent in general are $u \in \{-1,\ 0,\ +1\}$, which correspond to {"Toward *Base*", "Stay", "Away from *Base*"} respectively. However, the specific controls u_i available for the i^{th} agent depend on the agent's current location y_i and its remaining fuel f_i, according to the following rules:

$$u_i \in \begin{cases} \{-1,\ 0,\ +1\}, & \text{if } y_i = Y_C \\ \{-1,\ 0\}, & \text{if } y_i = Y_S \\ \{0,\ +1\}, & \text{if } y_i = Y_B \\ \{0\}, & \text{if } f_i = 0 \end{cases} \tag{5.23}$$

The total system action vector $\mathbf{u}$ for the decentralized formulation is simply $\mathbf{u} = u_i$, leaving the size of the action space $|\mathcal{A}| = (|u_i|)$.

5.3.3 State Transition Model P

The state transition model P captures the qualitative description of the dynamics of the state, given an action. As the state is divided into $\mathcal{S}_i$ and $\mathcal{S}_\Omega$, transitions for each, P_i and P_Ω, are given.

Local State Transitions, P_i

The model for agent location y_i are deterministic and are described by the following rules:

$$y_i(k+1) = \begin{cases} y_i(k), & \text{if: } f_i = 0 \\ Y_B, & \text{if: } y_i(k) = Y_B,\ u_i(k) = 0 \\ Y_B, & \text{if: } y_i(k) = Y_C,\ u_i(k) = -1 \\ Y_C, & \text{if: } y_i(k) = Y_C,\ u_i(k) = 0 \\ Y_C, & \text{if: } y_i(k) = Y_B,\ u_i(k) = +1 \\ Y_C, & \text{if: } y_i(k) = Y_S,\ u_i(k) = -1 \\ Y_S, & \text{if: } y_i(k) = Y_S,\ u_i(k) = 0 \\ Y_S, & \text{if: } y_i(k) = Y_C,\ u_i(k) = +1 \end{cases} \tag{5.24}$$

The dynamics for the fuel state f_i are stochastic with parameter p_f representing the probability of burning fuel at the nominal rate of one Δf per timestep. Specifically,

f_i evolves according to the following rules:

$$f_i(k+1) = \begin{cases} 0, & \text{if: } f_i(k) = 0 \\ F_{max}, & \text{if: } y_i(k) = Y_B \\ f_i(k) - \Delta f, & \text{w/ } \Pr(p_f) \text{ if } y_i(k) \neq Y_B \\ f_i(k) - 2\Delta f, & \text{w/ } \Pr(1 - p_f) \text{ if } y_i(k) \neq Y_B \end{cases} \tag{5.25}$$

The health state of each agent is also a stochastic model with parameters p_s and p_a representing the probability of a sensor failure, and actuator damage respectively. This health model evolves according to the following rules:

$$h_i(k+1) = \begin{cases} h_i(k), & \text{if } f_i = 0 \\ H_{nom}, & \text{if } y_i(k) = Y_B \\ H_{nom}, & \text{w/ } \Pr(1 - p_s - p_a) \text{ if } h_i(k) = H_{nom} \\ H_{sns}, & \text{w/ } \Pr(p_s) \text{ if } h_i(k) = H_{nom} \\ H_{act}, & \text{w/ } \Pr(p_a) \text{ if } h_i(k) = H_{nom} \end{cases} \tag{5.26}$$

Feature Transitions, P_Ω

An accurate description of the dynamics of the features describing agent i's teammates is critical for agent i to make intelligent, low-cost decisions. However, just as agent i relies on knowing each of its teammates' future states, each teammate also relies on knowing the future state of their teammates - including agent i's. In order to circumvent this problem, an iterative approach to determining P_Ω was implemented where an initial guess was made for P_Ω which was used in calculating policies for n GA-Dec-MMDPs. Using these policies, a look-up table that quantitatively describes the transition probabilities between all realizations of $(\mathcal{S}_\Omega \times \mathcal{A}_\Omega)$ was generated by evaluating corresponding state trajectories while running the PSM mission with $n = 5$ agents [87]. This process was repeated until the P_Ω converged to that shown in the upper-left of Figure 5-2. Transferring this approximation to cases with more agents

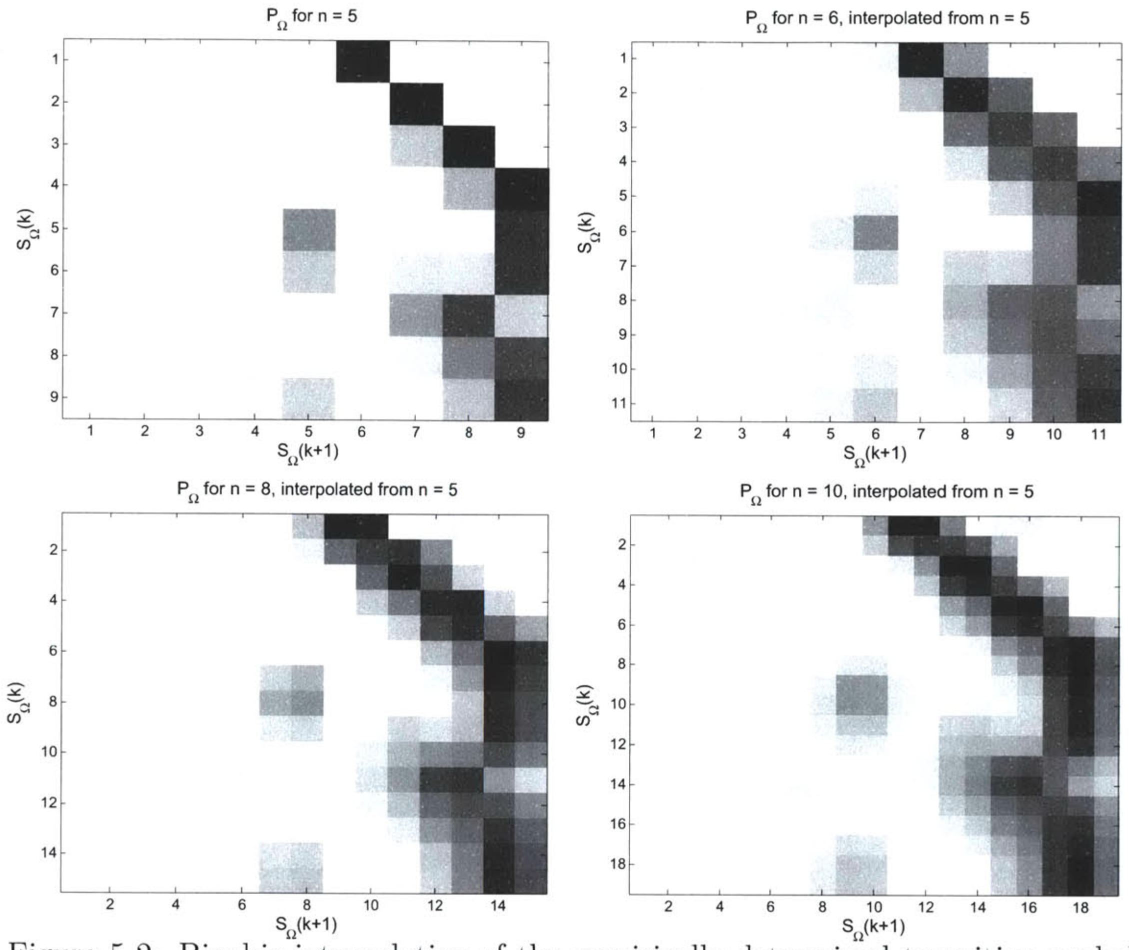


Figure 5-2: Bicubic interpolation of the empirically-determined transition probabilities for P_Ω from $n = 5$ (top-left) to $n = 10$ (bottom-right). Darker areas indicate higher probability and each row sums to 1.

(cases intractable for MMDP and even for Dec-MMDP) is not immediately clear. One approach is gleaned from the image processing literature, as scaling the empirical look-up table is essentially the same problem as zooming in on an image. Figure 5-2 visualizes the results of using bicubic interpolation to estimate the feature transition probabilities for cases where $n > 5$.

5.3.4 Cost Function g

The cost function $g(\mathbf{x}, \mathbf{u})$ in the decentralized case is set up to penalize any undesirable outcomes in the mission. However, the presence of approximations in forming $\mathcal{S}_\Omega$ remove some of the reward coupling between agents and cannot therefore "peek" into future possibilities the way the centralized problem can.

$$\begin{aligned} g(\mathbf{x}, \mathbf{u}) =& C_u n_u + C_{mot} n_{mot} + C_{sns} n_{sns} + \\ & C_s(n - 1 - n_s) + C_c\left(1 - \mathbf{I}_{[n_c > 0]}\right) + C_x n_x \end{aligned} \tag{5.27}$$

where $C_\star$ are costs associated with the following numbers: $n_u \in \{0,\ 1\}$ denotes agent movement, $n_{mot} \in \{0,\ 1\}$ indicates if the agent is in the task area with a degraded motor, $n_{sns} \in \{0,\ 1\}$ indicates if the agent is in the task area with failed sensor, $n_s \in \{0,\ n\}$ denotes the number of capable agents in the surveillance area, $n_c \in \{0,\ n\}$ denotes the number of agents in the communication area, and n_X is the number of agents that have run out of fuel (crashed).

5.3.5 Single-agent Policy $\pi^{n=1}$

Both Dec-MMDP and GA-Dec-MMDP formulations utilize a fixed policy, $\pi^{n=1}$, that is the result of formulating and solving a single-agent MDP where $\mathcal{S}$ is $\mathcal{S}_i$ from Section 5.3.1, $\mathcal{A}$ is identical to that formulated in Section 5.3.2, P is P_i from Section 5.3.3 and g is modified to remove the communication relay requirement.

5.4 Complexity Comparison of Mission Formulations

As the focus of this thesis is on problem formulation rather than on any particular solution approach, all solutions are computed using exact value-iteration, which is known to be $\mathcal{O}(|S|^2|A|)$ [16]. Because of this, we can use the term "computational complexity" in lieu of "state-space size". Tables 5.1 and 5.2 provide an idea of the connection between state-space size and the computation time required to initially solve the PSM mission when formulated as an MMDP, Dec-MMDP and GA-Dec-MMDP.

In the preceding sections, we formulated the PSM mission scenario as an MMDP, Dec-MMDP and GA-Dec-MMDP. Figure 5-3 graphically conceptualizes the relative

	MMDP	Dec-MMDP	GA-Dec-MMDP
# States	389,017	2,628	365
Computation Time	18 m 8 s	1.2 s	0.9 s

Table 5.1: Comparison of problem size and computation-time required for centralized and decentralized MDP-based multi-agent planners for a 3-agent persistent surveillance mission

	MMDP	Dec-MMDP	GA-Dec-MMDP
# States	28,398,241	15,768	511
Computation Time	$\approx$ 3 wks	2.8 s	1.1 s

Table 5.2: Comparison of problem size and computation-time required for centralized and decentralized MDP-based multi-agent planners for a 4-agent persistent surveillance mission

complexity of each of these formulations as the number of participating agents increases. Note, the y-axis of the figure is log-scale and shows the exponential growth in n of MMDPs and in $(n-1)$ of Dec-MMDPs. The GA-Dec-MMDP formulation however, remains linear in n. Also of interest is the black horizontal dashed line shown in Figure 5-3, which represents the approximate size of the PSM formulation whose solution can be re-computed online as parameter estimates change. That is, when bootstrapping a previous solution, formulations with roughly $200,000$ states are solvable on a timescale commensurate with the dynamics of the mission. Chapter 6 provides a performance comparison under both software and hardware flight experiments.

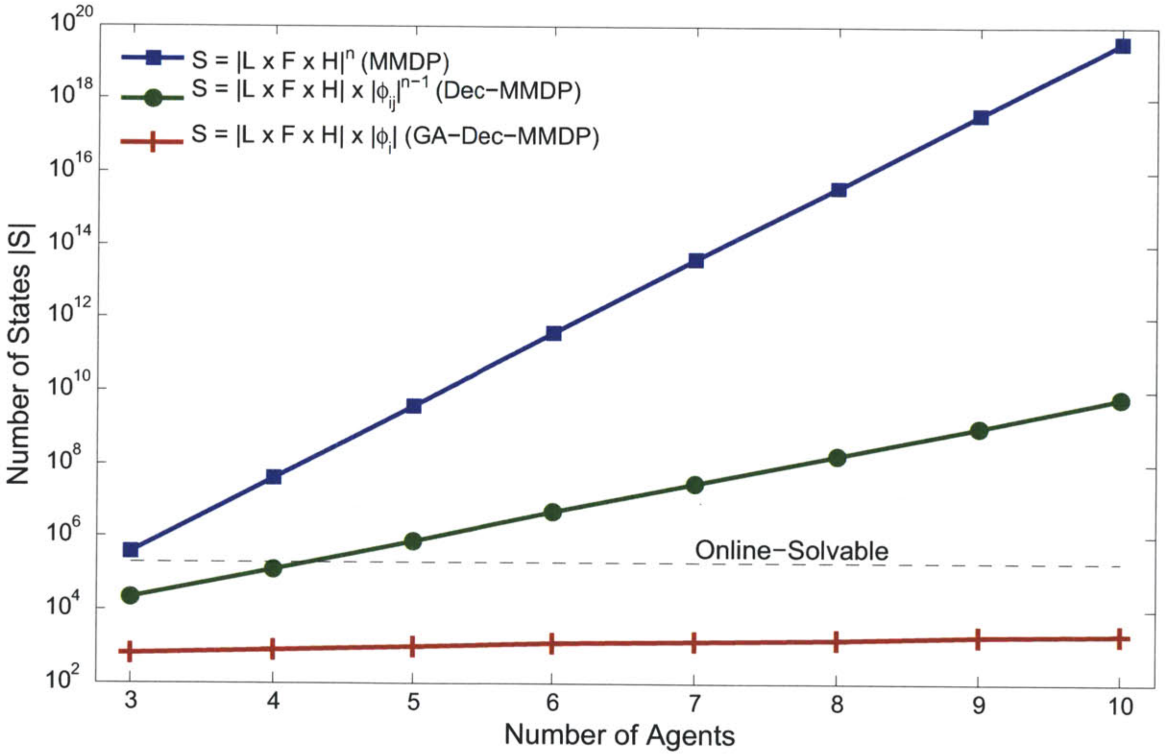


Figure 5-3: As the number of agents n increases, the resulting state space, and therefore also the computational complexity associated with calculating a solution to the PSM scenario, can grow exponentially and the choice of planning algorithm becomes critical. Note, the y-axis is log scale.

Chapter 6

Simulation and Experimental Results

The purpose of this chapter is to present experimental results from both software- and hardware-based flight experiments where the iCCA framework of Chapter 3 is combined with the approximate planning formulations of Chapter 4 under the PSM mission described in Chapter 5. In addition, this chapter provides the environmental details surrounding these experiments, such as the hardware developed in-house specifically for the purposes of this thesis and the PSM mission. We proceed as follows: Section 6.1 presents and discusses the iCCA framework as implemented for both hardware- and software-based experiments. Section 6.2 presents simulation results and details of the software-based simulation environment while Section 6.3 discusses the necessary elements of the hardware flight tests and presents the corresponding results.

6.1 Experimental Architecture

Figure 6-1 shows how the iCCA framework connects with the simulation and hardware experiments presented in the remainder of this chapter. The red arrows below and beside the *Mission Software* module represent a software switch that enables commands to be sent to either robotic agents in hardware or to simulated agents in

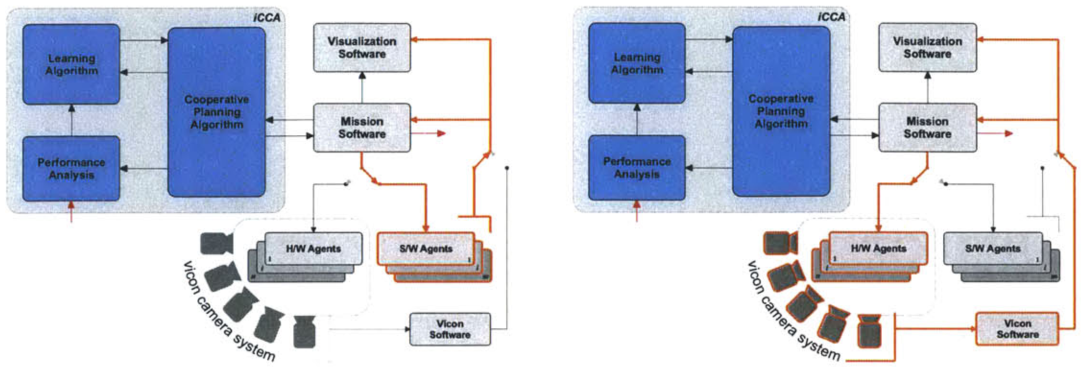


Figure 6-1: Depiction of the software and hardware loops used in generating simulation and flight test results. A software switch in the Mission Software (shown in red) toggles between flying simulated and actual hardware agents. Under simulation (left), a physics-based mathematical agent model propagates agent kinematic states. When flying in hardware (right), a Vicon motion capture system tracks and reports agent kinematic states.

software. The *Visualization Software* receives location and pose information (location, attitude, velocites, rates, etc...) about each participating agent either from the camera-based, indoor metrology system (see Section 6.3.3) or from a bank of software agent models. This information is graphically depicted alongside input commands within a 3D model of the mission environment to provide an excellent, interactive system for algorithm development and experimentation.

The *Mission Software* module manages the mission scenario throughout the duration of each experiment, including initialization and iCCA interaction. This module also receives position and pose information for each agent and uses this information to construct the "state" of the system in the format expected by iCCA's cooperative planner. This system state is then sent to iCCA, which queries the cooperative planner's policy for the "best" action.

This architecture was developed for the purposes of algorithm development and streamlining the transition to hardware flight experiments and is used in the simulation and hardware flight experiments that are detailed in the remainder of this chapter. It is important to note that the planning algorithms were intentionally made blind to whether hardware- or simulated-agents were being used. This allows the physical transition between software- and hardware-based tests to be realized by

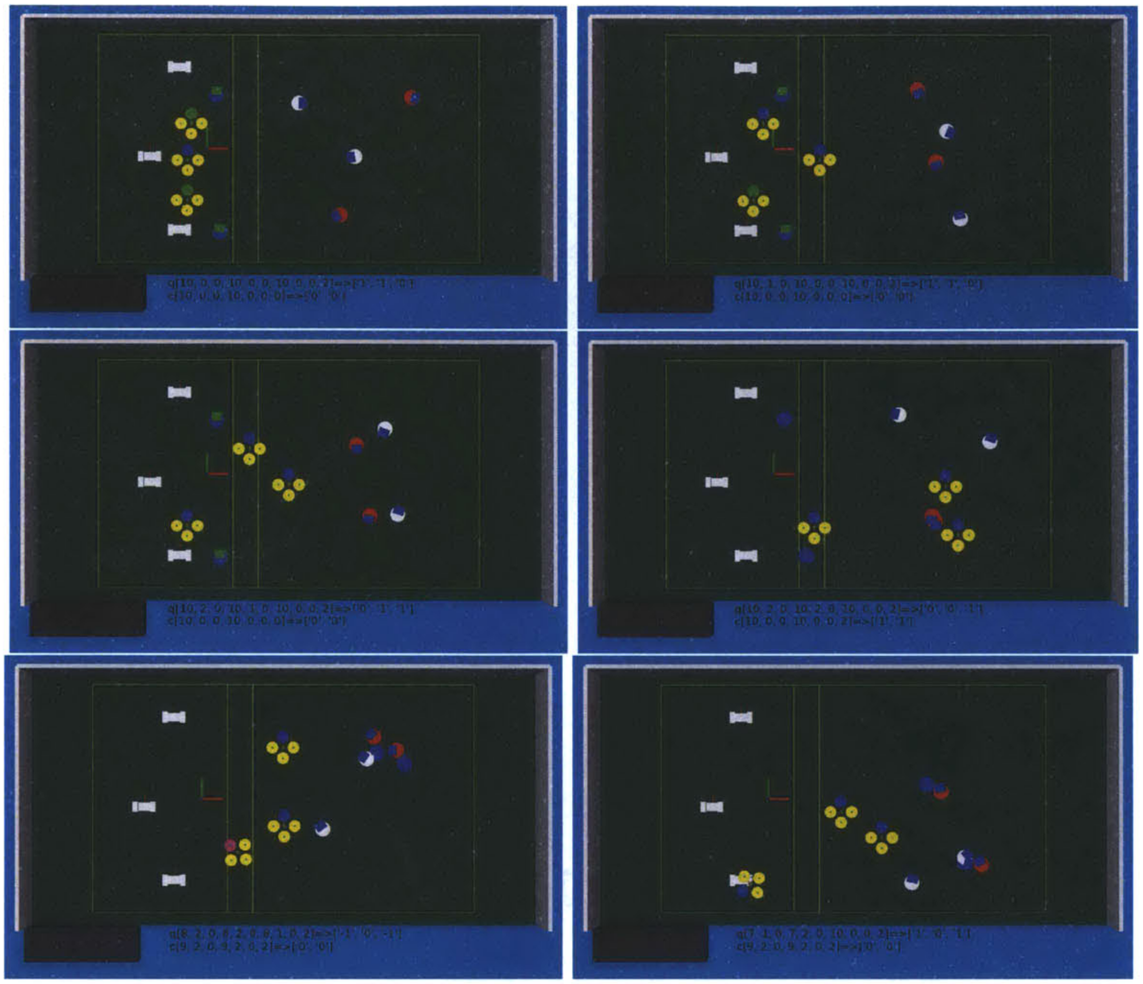
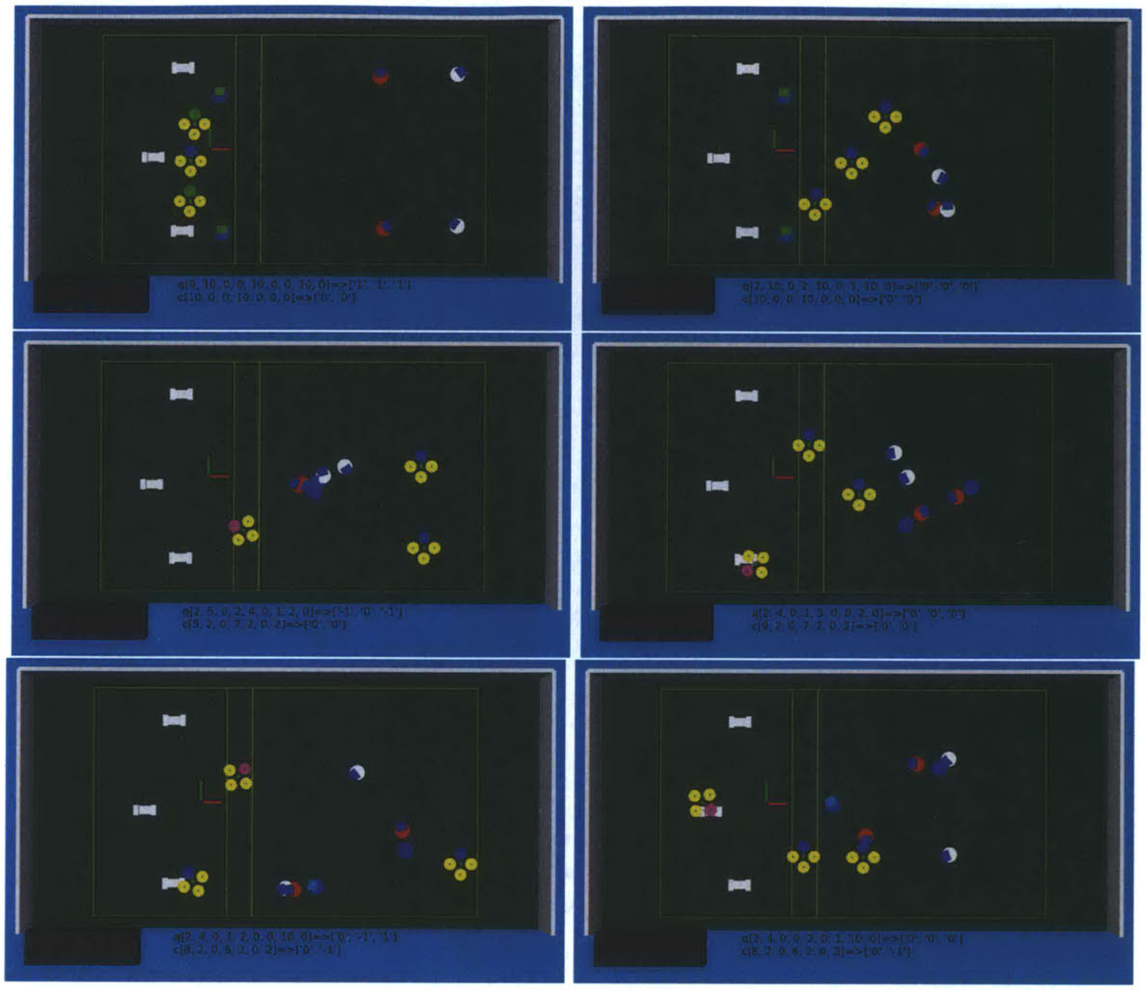


Figure 6-2: A simulated persistent surveillance mission (PSM) using the centralized multi-agent MDP (MMDP) formulation. The agents must coordinate to ensure no gaps in surveillance coverage while maintaining a communication relay.

toggling a boolean (software switch) in the common *Visualization Software*. This switch is shared with the *Mission Software* of Figure 6-1. Figures 6-2 and 6-3 are screenshots of the *Visualization Software* during simulated mission experiments.

6.2 Simulation Results

Figures 6-2 and 6-3 depict a series of screenshots from simulations running the persistent surveillance mission under centralized and Dec-MMDP policy formulations, respectively. In each of the series, the aerial agents (yellow) take-off from the base location, move toward the surveillance area and begin searching for targets (red). Upon



```
a[0, 10, 0, 0, 10, 0, 0, 10, 0]=>['1', '1', '1']
c[10, 0, 0, 10, 0, 0, 0]=>['0', '0']
```

```
a[2, 10, 0, 2, 10, 0, 1, 10, 0]=>['0', '0', '0']
c[10, 0, 0, 10, 0, 0, 0]=>['0', '0']
```

```
a[2, 5, 0, 2, 4, 0, 1, 2, 0]=>['-1', '0', '-1']
c[9, 2, 0, 7, 2, 0, 2]=>['0', '0']
```

```
a[2, 4, 0, 1, 3, 0, 0, 2, 0]=>['0', '0', '0']
c[9, 2, 0, 7, 2, 0, 2]=>['0', '0']
```

```
a[2, 4, 0, 1, 2, 0, 0, 10, 0]=>['0', '-1', '1']
c[8, 2, 0, 6, 2, 0, 2]=>['0', '-1']
```

```
a[2, 4, 0, 0, 2, 0, 1, 10, 0]=>['0', '0', '0']
c[8, 2, 0, 6, 2, 0, 2]=>['0', '-1']
```

Figure 6-3: A simulated persistent surveillance mission (PSM) using the decentralized multi-agent MDP (Dec-MMDP) formulation. The agents must coordinate to ensure no gaps in surveillance coverage while maintaining a communication relay.

detection, ground agents (blue/green) assume tracking duty while the aerial agents continue searching. The white/blue ground agents are "civilians" and the ground agents must avoid collisions with them. Throughout the mission, aerial and ground agents alike must occasionally return to base to refuel. The agents must coordinate with each other when doing so to ensure that no gaps in the coverage of the surveillance region occur and that whenever one or more agents are in the surveillance area, there is another providing a communication relay with the base.

Algorithm 1 State-dependent Heuristic

```
Input: agents
Output: u
Initialize: $ii$ = commAgent = -1
for a in agents:
  $ii$ += 1
  if a.pos==BASE and a.fuel==Fmax:
    $u[ii] = 1$
  elif a.pos==COMM and a.fuel>2:
    if commAgent==ii or commAgent<0:
      $u[ii] = 0$
      commAgent = $ii$
    else :
      $u[ii] = 1$
  elif a.pos==COMM:
    $u[ii] = -1$
    if commAgent==$ii$:
      commAgent = -1
  elif a.pos==SURV and a.fuel < 4:
    $u[ii] = -1$
  elif a.pos==SURV:
    if commAgent<0:
      $u[ii] = -1$
      commAgent = $ii$
    else :
      $u[ii] = 0$
```

6.2.1 Cooperative Multi-Agent Planners

Monte Carlo style simulations of the full, stochastic PSM scenario were run in this environment for several planning approaches, including: A non-cooperative approach; a state-dependent heuristic; a fully centralized MMDP (for the case of $n = 3$ only); a Dec-MMDP as formulated in Section 5.2; and a GA-Dec-MMDP, as formulated in Section 5.3.

First, the non-cooperative approach simulates each agent acting for itself only and is included as benchmark to allow a more complete understanding of the costs incurred by the cooperative planners. Second, Algorithm 1 shows the state-dependent heuristic that represents the type of behavior a human operator might encode for an unmanned agent in the PSM mission. Essentially, this heuristic tells each agent to "Go to the surveillance area until a fixed fuel-level is reached, then go back to base to refuel. Secondly, if along the way, an agent finds itself in the communication area with sufficient fuel then it will become the comm-relay if no other agent is currently fulling the role. It will perform this task until a fixed fuel-level and then go refuel.

The other three planners up for comparison are MDP-based, and as this research focuses on problem formulation rather than on particular solution approaches, all solutions were computed using exact value-iteration, which is known to be $\mathcal{O}(|S|^2|A|)$ [16]. Because of this, we can use the term "computational complexity" in lieu of "state-space size".

For the PSM scenario simulations, the following parameters were used: $\Delta f = 1$, $F_{max} = 10$, $p_f = 0.50$, $p_a = 0.05$, $p_s = 0.10$. The simulations were run on a 64-bit quad-core Intel Xeon 3.33 GHz CPU running Ubuntu 11.04 with GCC 4.5 and 12 GB of RAM. For the cases where n is greater than three, the MMDP approach is simply not tractable, having more than 28 million states when $n = 4$. Therefore, comparisons beyond $n = 3$ do not include MMDP results. Similarly, the Dec-MMDP approach remains computationally tractable until $n = 5$ agents, beyond which it too becomes intractable. The GA-Dec-MMDP approach however, remains tractable through $n = 10$. We therefore compare GA-Dec-MMDP with the non-cooperative and state-dependent heuristic approaches for cases when $n > 5$.

After ensuring identical starting conditions, each planner was simulated for 500 steps, 50 times each, logging the joint state trajectories for each system. Using these state histories, the average cumulative return for each planner was calculated using an evaluation cost function described by

$$g(\mathbf{x}) = C_c\left(1 - \mathbf{I}_{[n_c>0]}\right) + C_s(n - 1 - n_s) \tag{6.1}$$

where n_s is the number of capable agents in the surveillance area and n_c is the number of agents in the communication area. Figure 6-4 compares the scores of the different planners for the case of three agents ($n = 3$) in the persistent surveillance mission.

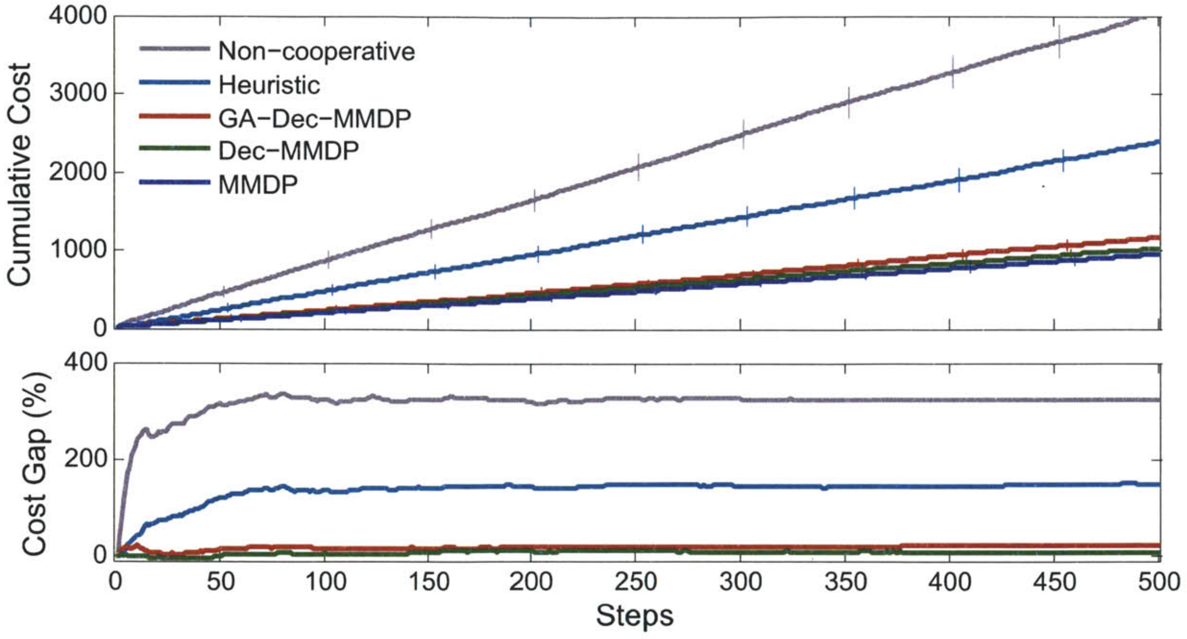


Figure 6-4: Comparison of cumulative costs over a 500 step stochastic PSM mission for the case of three agents ($n = 3$). As expected, the non-cooperative solution scores poorly while the MDP-based solutions provide the lowest cost solutions.

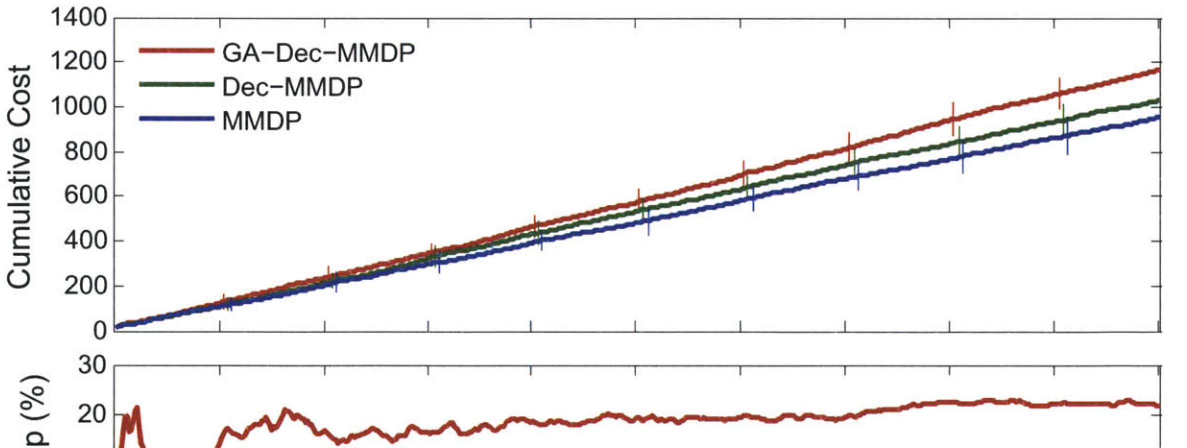


Figure 6-5: Removing the non-cooperative and heuristic-based approaches allows a clearer comparison of the cumulative costs over a 500 step mission for the case of three agents between the MDP-based methods only. As seen, Dec-MMDP results in roughly 10% higher cost than the MMDP while GA-Dec-MMDP yields approximately 20% higher cost.

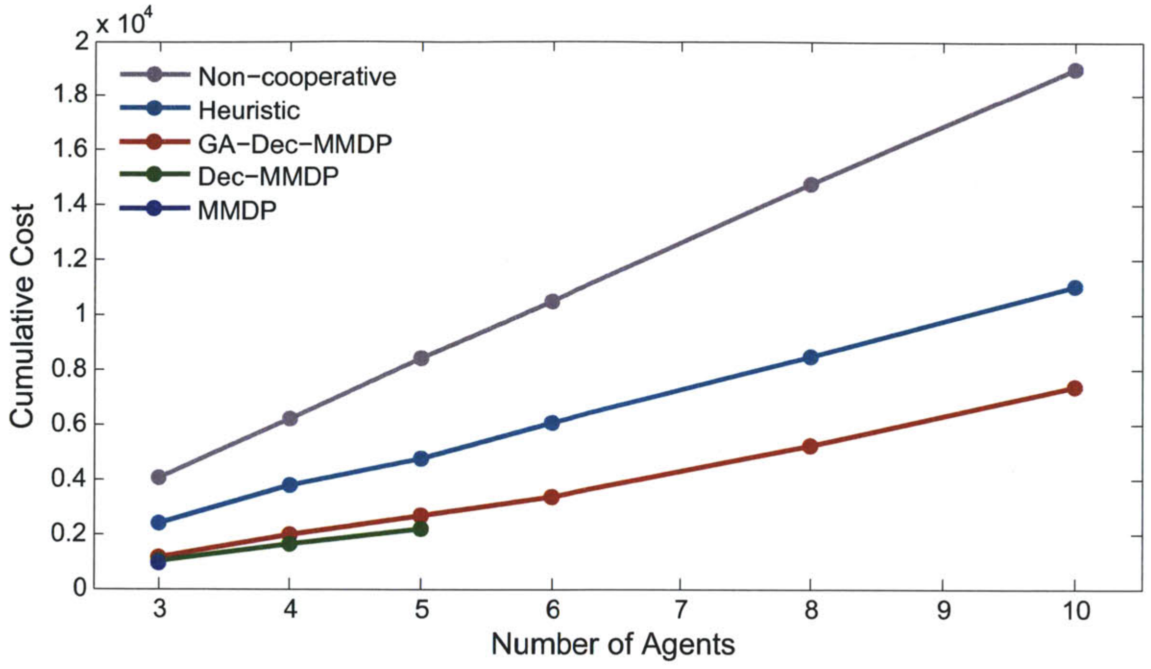


Figure 6-6: Comparison of resulting cumulative cost after a 500 step stochastic PSM mission as a function of the number of participating agents. GA-Dec-MMDP (red) remains the lowest cost tractable solution through $n = 10$ agents.

As expected, the MMDP results in the minimum cost solution, while the Dec-MMDP and GA-Dec-MMDP approximations are within 10% and 20% respectively. As the non-cooperative and state-dependent heuristic approaches result in much higher cost, and Figure 6-5 removes them for a closer look at the MDP-based strategies alone.

Scaling up now to $n = 10$, Figure 6-6 shows how the cumulative cost scales with the number of participating agents. While we cannot compare with the centralized MMDP or Dec-MMDP in these cases, the main point of this figure is to show that the cumulative cost of the GA-Dec-MMDP approximation consistently results in a much lower cost than the heuristic approach.

6.2.2 Learning and Performance Analysis

The previous section compares the performance of the cooperative planning module within iCCA for various formulations of the PSM scenario. In this section, we wrap these planners within the iCCA framework and show the results of the learning and performance analysis elements of iCCA for the complete, simulated system.

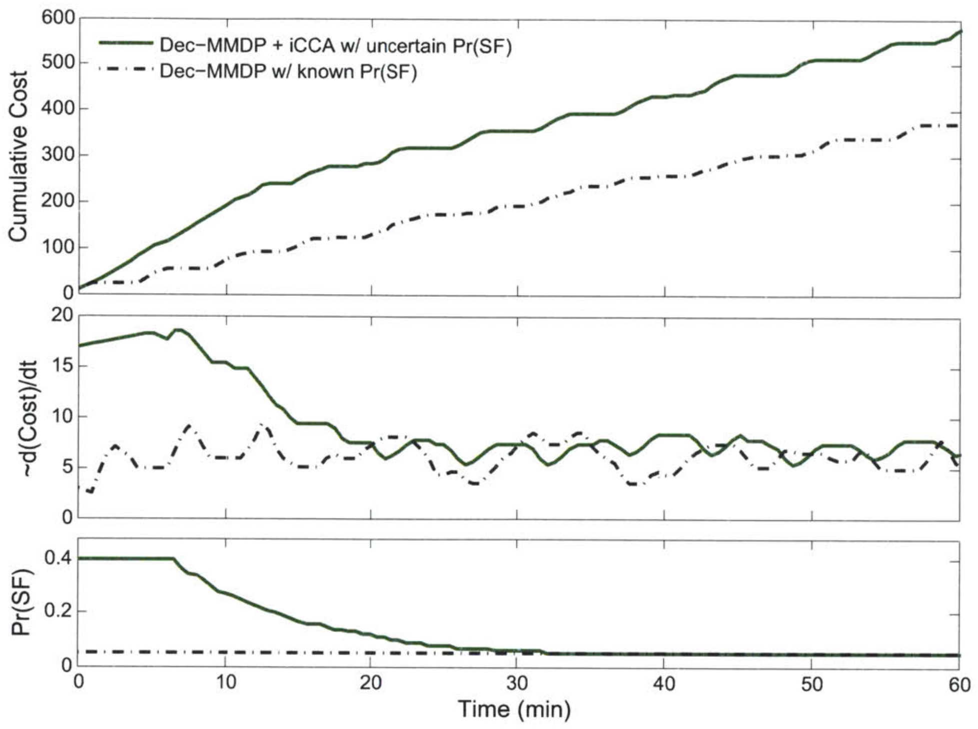


Figure 6-7: Results of formulating the persistent surveillance mission as a Dec-MMDP and implementing it in the hardware simulation environment portrayed in Figure 6-1 (left). On the top, accumulated cost is shown (lower is better) as the mission is carried out. In the middle, a filtered piecewise derivative of the top subplot provides a notion of how fast costs are being incurred. Finally, the lower subplot shows an agent's estimate of the probability of experiencing a sensor failure, which is updated over time by the learning algorithm. Solutions to the cooperative planner formulation are generated online as the uncertainty around the model parameter decreases.

The PSM scenario as formulated contains a number of uncertain parameters, to which the system performance is, to varying degrees, sensitive. For instance, there is a non-zero probability that an agent's sensor may fail during any state transition not involving the base area. The likelihood of this happening affects the policy of the associated cooperative planner. The policy, in turn, affects the performance of the system as it dictates decision-making. This relationship is shown in Figure 6-7.

Figure 6-7 consists of three subplots. The top plot shows the sum of the accumulated cost incurred by the cooperative planner for each agent in the mission thus, lower is better. The middle subplot is a filtered piecewise derivative of the top subplot and provides a notion of how fast costs are being incurred as the mission is carried out.

The bottom plot shows a single agent's estimate of the probability of experiencing a sensor failure, which is updated by the learning algorithm as the mission is carried out. At the start, when the estimated probability of sensor failure is high (bottom subplot), costs are incurred at a higher rate (green, solid) than when the parameter is known (black, dashed), as shown in the top and middle subplots. This is expected behavior since intelligent cooperation between agents relies on knowledge of sensor reliability, with perfect reliability leading to the lowest cost cooperative plans. The bottom subplot of Figure 6-7 shows how the probability of sensor failure parameter is initially high, and is learned through experience and interactions. As the parameter estimate (green, solid) approaches its true value (black, dashed) the rate of cost accrual decreases to eventually match the rate of cost accrued when the parameter is known. This result demonstrates the desired interaction between the planning and learning algorithms.

6.3 Hardware Setup and Results

There are three fundamental categories of hardware when flight-testing persistent, multi-agent missions indoors: mobile robotic agents, automated maintenance, and a fast and accurate global metrology system. This section discusses each of these categories and details the specific components used in the persistent surveillance mission scenario.

6.3.1 Mobile Robotic Agents

Four different agent platforms are used in the PSM scenario consisting of an aerial agent (a quadrotor UAV) and three variations of ground-based (UGV) agents. Referring to Figure 6-8, the quadrotors (one shown in upper-left) and the blue, camera-equipped UGVs (one shown in upper-right) comprise the team of agents planning the persistent mission. In addition, the yellow and green colored UGVs (lower left) are the targets that need to be discovered in the surveillance area while the white UGVs (lower right) represent civilians and are to be avoided.

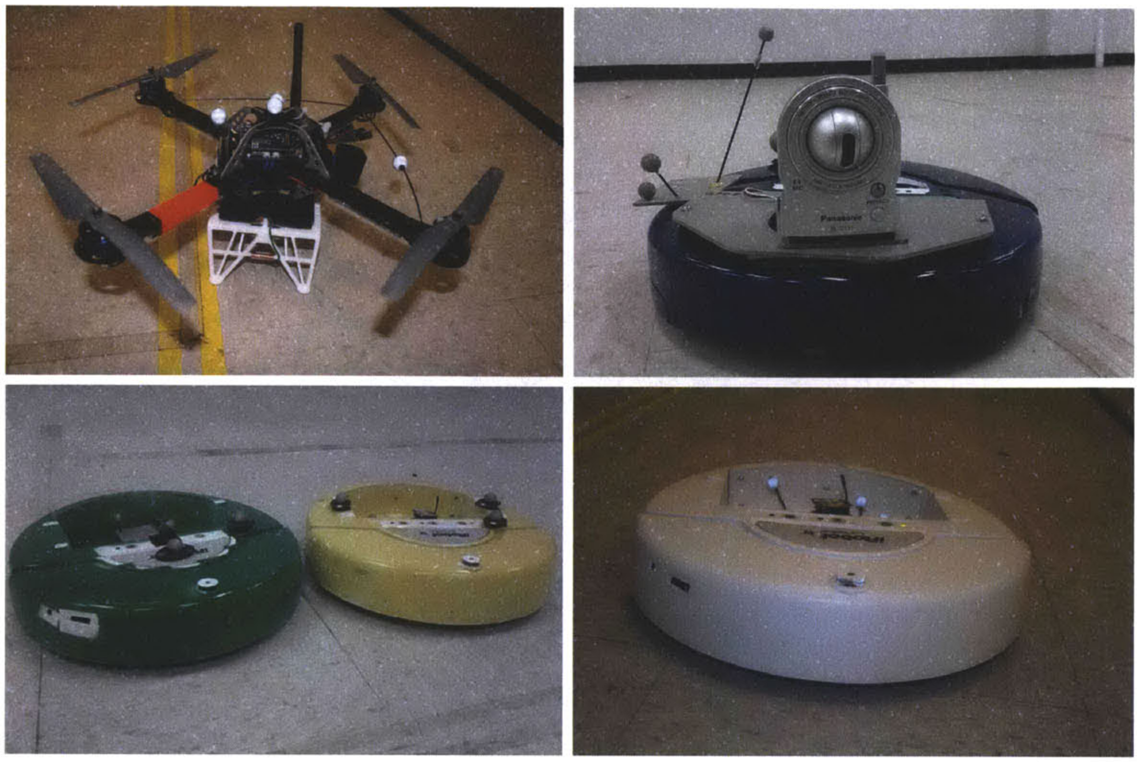


Figure 6-8: Four agent platforms used in flight experiments: Team UAV (top left), Team UGV (top right), Target UGV (bottom left), and a Civilian UGV (bottom right).

Also visible in Figure 6-8, the quadrotor UAVs were equipped with a wifi-enabled webcam and a recharge-capable base. The webcam provides a real-time capability for detecting the target UGVs while the base allows the quadrotor to lock into the change/charge station for refueling. In addition, team member UGVs are also equipped with onboard cameras for real-time detection of the target UGVs amongst the "civilians". Commonly available color detection algorithms were run to provide detection and localization of the target UGVs once seen in the camera image.

In addition, to answer the need for inexpensive, autonomous, aerial mobile robots, we also developed an in-house quadrotor platform for use in PSM and related flight experiments. The quadrotor, pictured in Figure 6-9, is built on a carbon-fiber and foam sandwich plate frame with brushless motors, electronic speed controllers [51], and an off-the-shelf autopilot board with accelerometers, gyros and a pic-based microcontroller[53]. The firmware for the autopilot was also developed in-house to close the roll and pitch

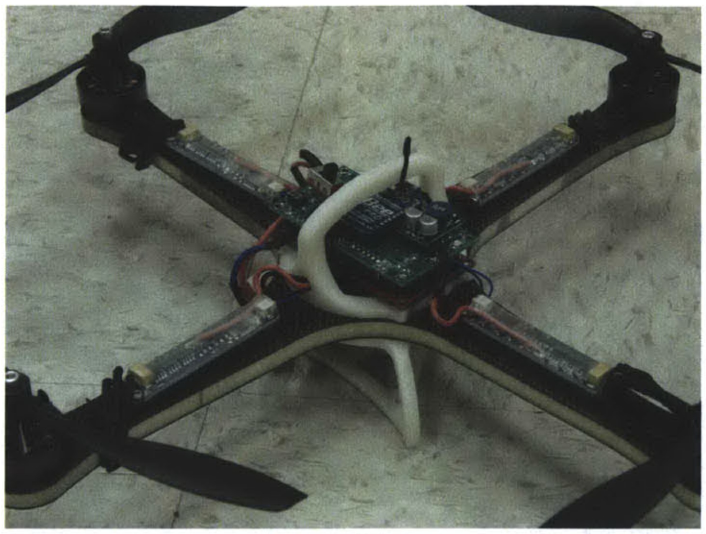


Figure 6-9: Quadrotor helicopter built in-house to answer our need for an inexpensive, autonomous, aerial mobile robot.

loops and yaw rate loop onboard for stable, hovering flight. As a bonus, the electronic speed controllers provide feedback on how much current the corresponding motor is drawing as well as its approximate temperature. These readings are useful in generating a snapshot of the vehicle's health in real-time.

6.3.2 Automated Robot Maintenance

As missions involving UAVs lengthen in duration, frequent and robust refueling becomes critical to overall success. In the research setting, refueling typically means charging or swapping onboard batteries. Most recently, the use of a high energy laser beam to charge the battery during flight has been introduced[75]. Automating and/or streamlining the recharging procedure has been the topic of much previous work [7, 28, 29, 45, 46, 59, 98, 104, 107–109]. However, waiting for a battery to properly recharge can be very time-consuming, causing delays in the overall mission. Also, "cold" battery swapping techniques require a complete shutdown of the vehicle's onboard electronics as the spent battery is swapped for a new one. This adds further delay and a potential for losing onboard data and state information.

As slow recharge times and cold battery swaps are undesirable, this section introduces an automated, portable landing platform capable of "hot" swapping batteries such that the UAV remains powered up throughout the process. The automated station holds a buffer of seven batteries in a novel dual-drum structure that enables time-efficient swapping. Each drum consists of four battery bays, each of which can be connected to a charger for proper battery maintenance and charging. The hot-swap capability, in combination with local recharging and a large battery capacity allow this platform to refuel multiple UAVs for long-duration and persistent missions with minimal delays and without vehicle shutdowns.

Under a nominal payload (including a 1350mAh 12.6 V battery and USB camera), the quadrotors detailed in Section 6.3.1 above have a conservative flight time of about 7 minutes. The change/charge station can hold seven additional batteries, which enables a flight time of nearly an hour without utilizing the station's recharge capabilities. However, enabling the local recharge capability removes the upper limit on the operational flight time (when servicing a single vehicle), as a spent battery typically needs less time to charge than the flight time provided by the other batteries in the station (assuming that all seven batteries are fully-charged initially).

Figure 6-10 shows the complete system for automated battery maintenance. As seen, a retro-fitted quadrotor sits in a sloped landing plate and is held securely in place with two locking arms. The quadrotor lands between two rotating drums, each consisting of four battery bays which can carry (and optionally recharge) a single battery. The system as a whole provides a fully automated battery change/charge capability which can eliminate the need for a human operator to manually swap or recharge the batteries on the vehicle. Such a system offers an efficient and portable solution to the battery recharge problem.

The battery swapping process consists of the following 6 steps, which are also illustrated in Figure 6-11:

1. A quadrotor lands on the center plate, turns off its motors, and sends a message to the off-board control software indicating that the landing is complete.

2. The off-board control software commands the change station to lock/align the

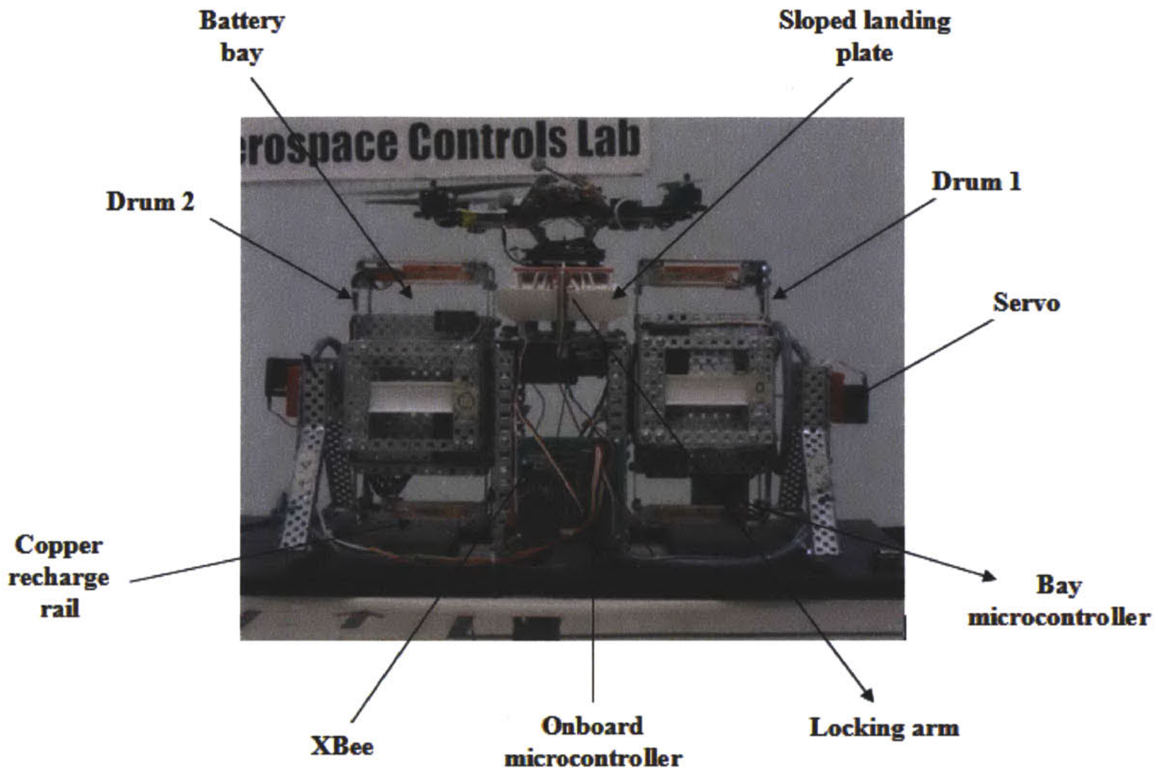


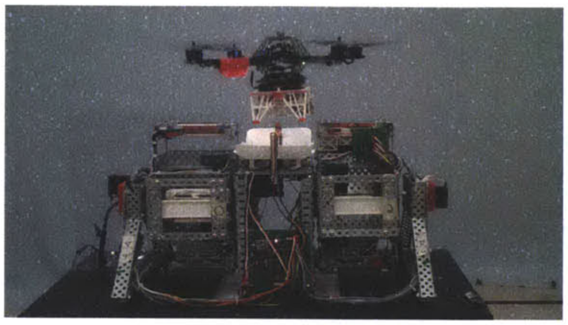
Figure 6-10: The change/charge station is outfitted with 11 motors, 2 servos, 9 limit switch sensors, 2 rotational encoders, 3 custom PCBs, 3 microprocessors, an XBee wireless modem and optionally 8 battery chargers (not shown).

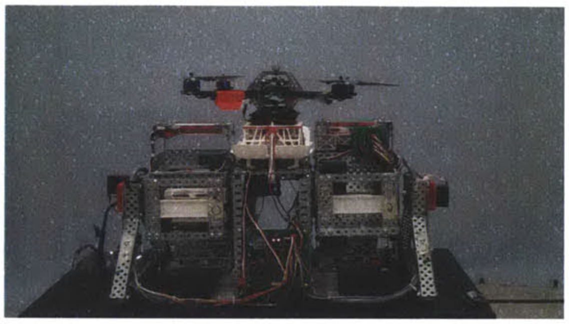
quadrotor using its servo arms. A proper lock/alignment will activate the center switch, thereby notifying the off-board software.

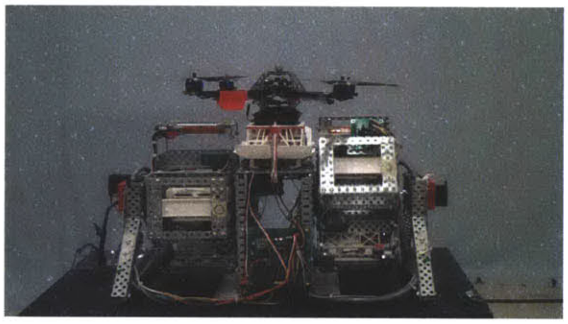
3. With the quadrotor properly aligned and locked, the control software then scans the voltage levels of each bay and determines which bay is empty and which contains the battery with the highest voltage level. Commands are then sent to the change station to rotate the drums such that these two bays are aligned with the center section.

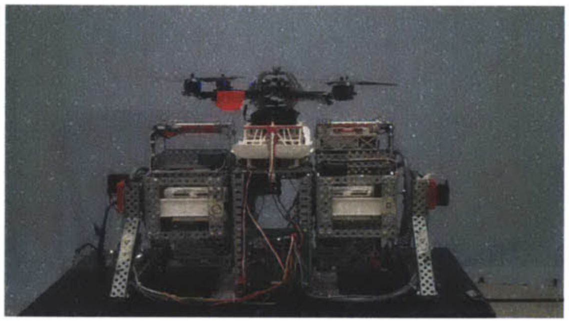
4. Once the drums are properly rotated, the control software sends commands to the change station to eject the charged battery from its bay while moving the spent battery out from under the quadrotor and toward the empty bay.

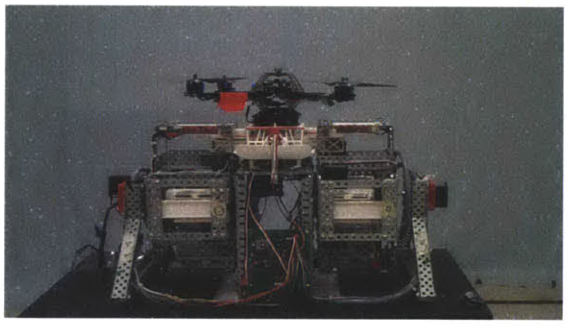
5. As the charged battery moves into its proper place in the receiver under the

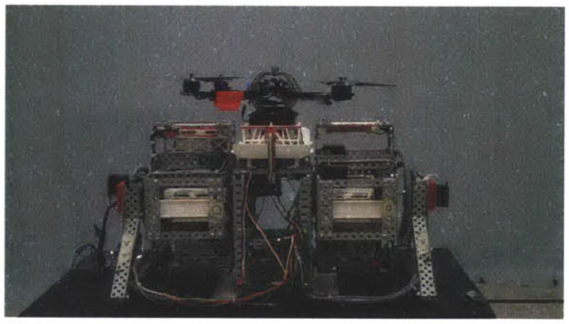

(a) Quadrotor hovers over pad and descends to land

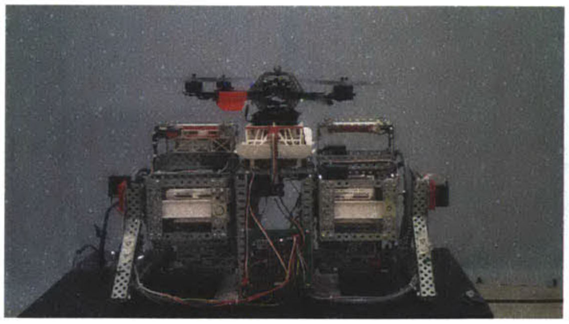

(b) Quadrotor clamped to pad with shore power


(c) New battery chosen and right drum rotates


(d) Drums now aligned


(e) Replacement battery pushed into place (right to left), moving the old battery into the left drum


(f) After the battery swap is finished, the clamps are released


(g) Quadrotor motors restarted

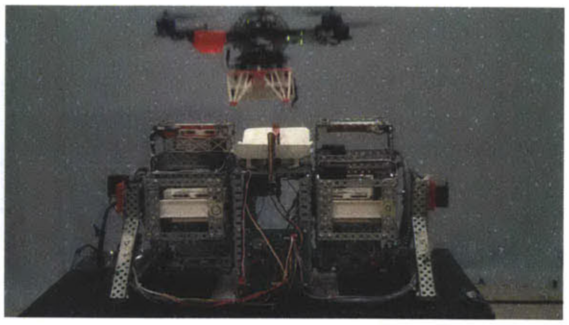

(h) Quadrotor takes off and returns to mission

Figure 6-11: One battery swap sequence from a multi-swap mission.

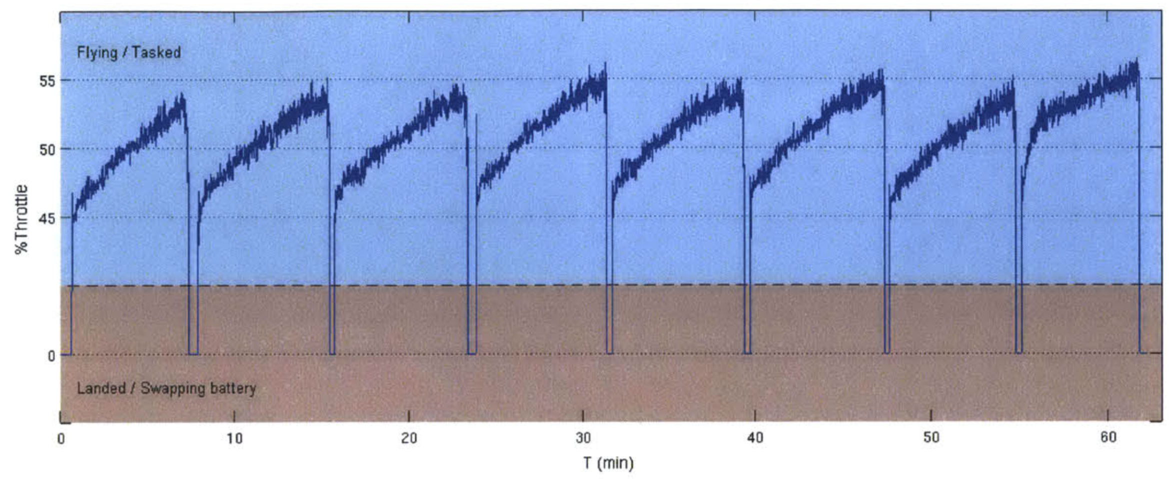


Figure 6-12: Plot of the collective control input to a quadrotor helicopter during a second mission lasting > 60 minutes. As seen, battery swaps require less than 30 s between steps 1 and 6 of the swap sequence.

quadrotor, it activates the center switch and the center motor is stopped and the locking arms are released.

6. Upon activation of the center switch, the control software then sends a message to the quadrotor, clearing it for take-off. If this sequence is part of a larger mission, it is then up to the mission manager to decide when/if the quadrotor will actually take off.

Figure 6-12 shows the collective control input as it changes for a single quadrotor during a long-duration mission. In this case, the quadrotor swapped its battery seven times during a mission of over 60 minutes. This graph shows how the automated battery changing concept actually enables significantly longer duration missions (potentially indefinite with the inclusion of the charging mechanism) and in this case resulting in only 4.2% down time.

6.3.3 Indoor Metrology

Flight tests and algorithm demonstrations were carried out in two physical locations: The RAVEN facility at MIT's Aerospace Controls Lab and the VSTL at Boeing Research and Technology in Seattle, WA.

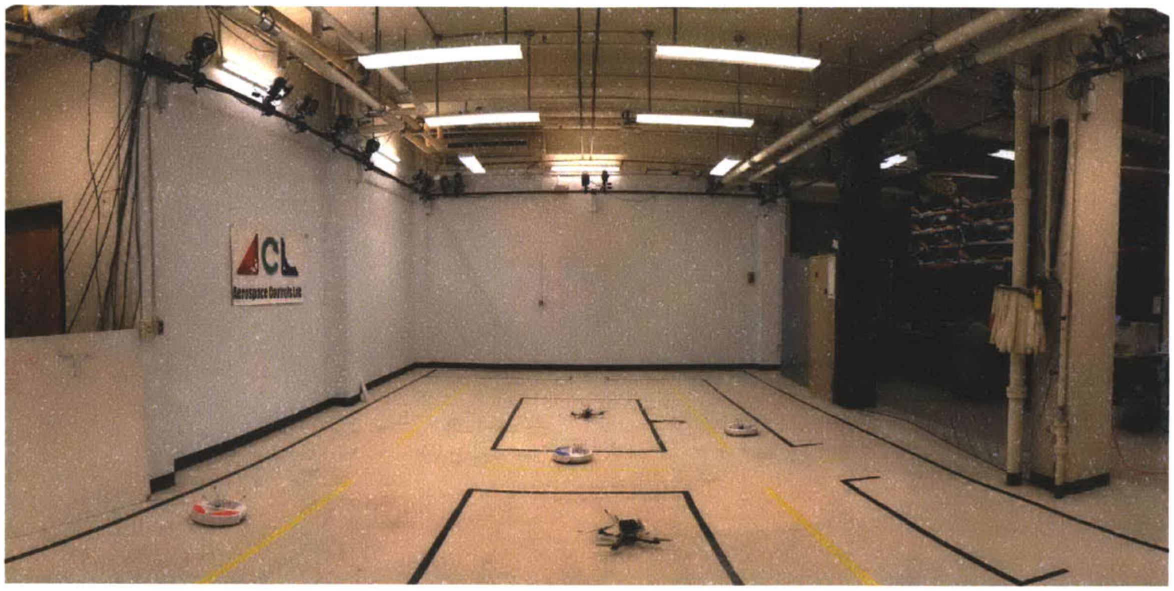


Figure 6-13: The Real-time Autonomous Vehicle test ENvironment (RAVEN) in the Aerospace Controls Lab at MIT

Figure 6-13 shows the general layout of the MIT RAVEN facility. The Realtime indoor Autonomous Vehicle test ENvironment (RAVEN) enables rapid prototyping and testing of a variety of unmanned vehicle technologies, such as adaptive flight control, automated UAV recharging, autonomous UAV air combat, and coordinated multi-vehicle search and track missions, in a controlled, indoor flight test volume. RAVEN utilizes a camera-based motion capture system to simultaneously track multiple air- and ground-based vehicles, and provide highly accurate position and orientation information about these vehicles in real-time. This information is then distributed to a group of command and control computers responsible for managing the autonomous execution of the mission.

Boeing Research and Technology has developed the Vehicle Swarm Technology Laboratory (VSTL), an environment for testing a variety of vehicles in an indoor, controlled environment [31]. VSTL is capable of simultaneously supporting a large number of both air and ground vehicles, thus providing a significant advantage over traditional flight test methods in terms of flight hours logged. As seen in Figure 6-14, the primary components of the VSTL are: 1) A camera-based motion capture system for reference positions, velocities, attitudes and attitude rates; 2) A cluster of off-

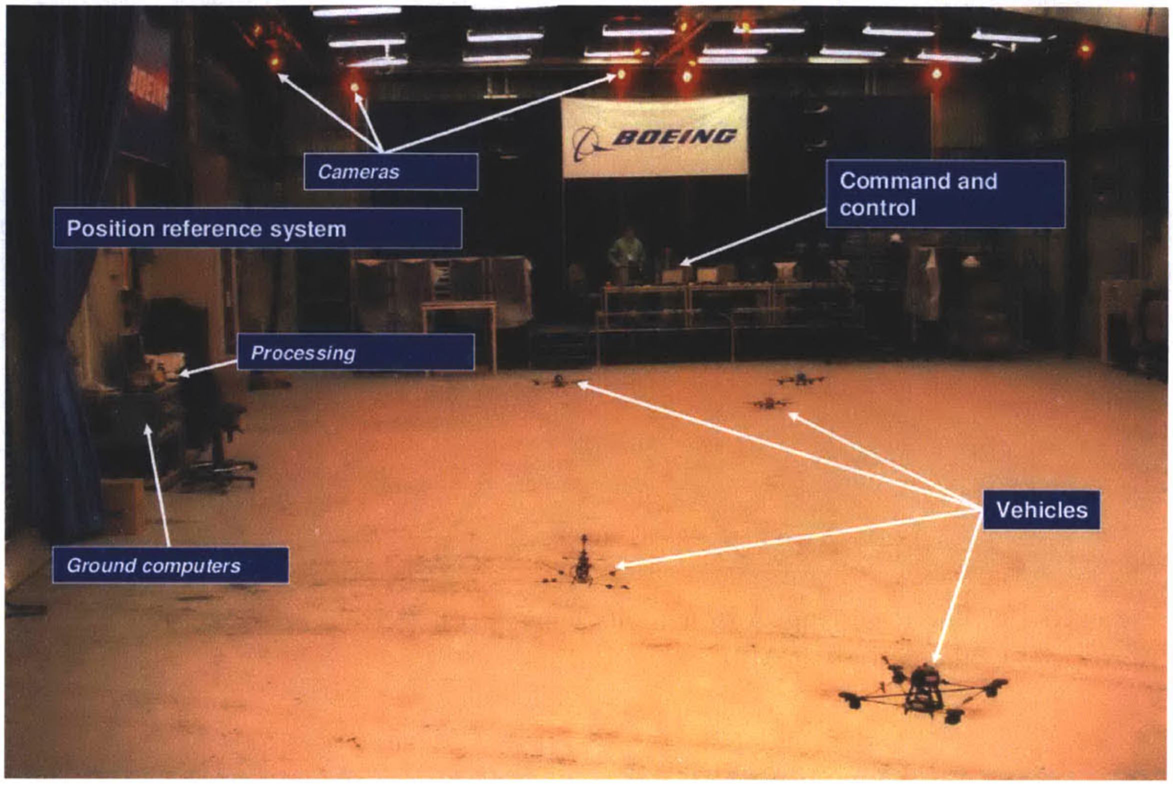


Figure 6-14: The Boeing Vehicle Swarm Technology Laboratory (VSTL), a state-of-the-art rapid prototyping indoor flight testing facility [31]

board computers for processing the reference data and calculating control inputs; 3) Operator interface software for providing high-level commands to individual and/or teams of agents. These components are networked within a systematic, modular architecture to support rapid development and prototyping of multi-agent algorithms [31].

6.3.4 Flight-test Results

For the flight tests, three quadrotor helicopters and two ground vehicles were placed in a persistent surveillance mission as cooperating, but independent agents. A Dec-MMDP cooperative planner and an exponential-based learning algorithm were coupled through the uncertainty of sensor failure. As the learning algorithm worked to converge on the true value of the parameter, the planner utilized the updated estimates and was able to improve subsequent plans as the parameter uncertainty was

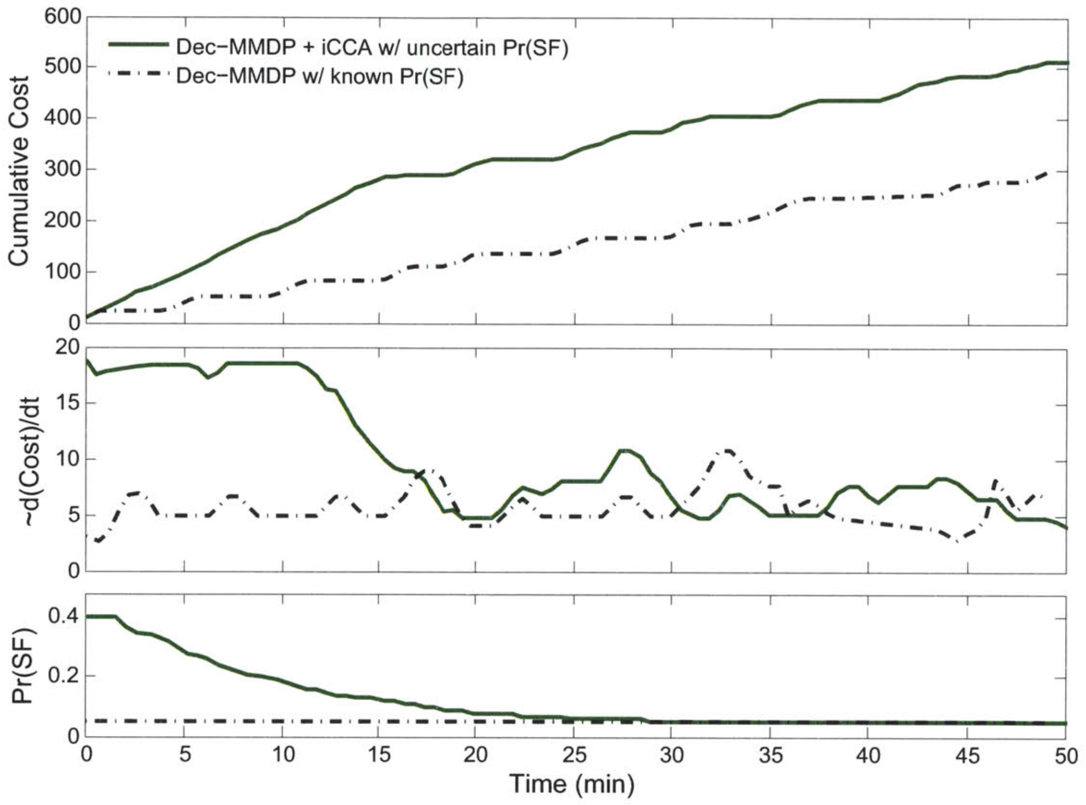


Figure 6-15: Results of formulating the persistent surveillance mission as a Dec-MMDP and implementing it in a real-time, actual flight test scenario where solutions are generated online as the uncertainty around model parameters decreases as they are learned using iFDD. On top, accumulated cost is shown (lower is better) as the mission is carried out. In the middle, a filtered piecewise derivative of the top subplot provides a notion of roughly how fast costs are being incurred. The lower subplot shows an agent's estimate of the probability of experiencing a sensor failure, which is updated over time by the learning algorithm. Solutions to the cooperative planner formulation are generated online as the uncertainty around the model parameter decreases, thus improving subsequent plans.

decreased. The results in Figure 6-15 consist of three subplots. The top plot shows the sum of the accumulated cost incurred by the cooperative planner for each agent in the mission, thus lower is better and the slope of the lines in this plot represent the rate at which costs are accrued. The middle subplot is a filtered piecewise derivative of the top subplot and provides a notion of how fast costs are being incurred. The bottom plot shows a single agent's estimate of the probability of experiencing a sensor failure.

As expected, these results are in significant agreement with the simulation results previously shown in Figure 6-7. As in the simulated tests, these flight results demonstrate the desired interaction between the planning and learning algorithms and the system's ability to learn from experience and improve the overall performance over time.

Chapter 7

Conclusion

This thesis consisted of three focus areas. The first was the integration of multi-agent planning algorithms with online adaptation, or learning, algorithms within a common architecture. The motivation for such an architecture was that, in many applications, the basic form of models internal to the planner are known, while their associated parameters are not. As a result, the uncertainty around these parameters may lead to significant suboptimal performance in the system. In addition, these parameters may be time-varying, which introduces further sub-optimalities, especially if the planning problem is too large to be solved online. To address these issues, an architecture for continuous online planning, learning and replanning was developed that combines a generic multi-agent planner with parameter learning and performance evaluation algorithms. This thesis contributed the following:

- A template architecture was developed to enable the integration of multi-agent cooperative control techniques with online learning algorithms and performance evaluation metrics. We called the result the *intelligent Cooperative Control Architecture*, or iCCA. As a modular template, iCCA permitted the use of a variety of multi-agent planning algorithms, learning methods and evaluation functions.

- The iCCA template was instantiated under multiple combinations of planner, learner and evaluation components and was implemented in both simulation and flight-test environments. Planner types included MDPs and a market-based iterative auction algorithm (CBBA). Under the learning component, techniques such as direct adaptive control, maximum likelihood estimation and reinforcement learning were included.

A second focus of this thesis was the development of approximation algorithms for the decentralized control of multiple agents in a cooperative environment. The motivation for such approximations was primarily to avoid the exponential explosion of required calculcations when considering the combinatorial coupling between participating agents under an MDP-based planner. This thesis addressed this challenge by altering the problem formulation slightly to allow each agent to approximate its teammates while maintaining a full-fidelity model of itself. Thus, when viewed collectively, the team contained a full model for each agent while the problem size for each agent remained small enough that a solution was shown to be computable within the timescale of the planning horizon (typically seconds, or minutes). We called our approach a *decentralized multi-agent Markov decision process (Dec-MMDP)* to emphasize the connection with the well-studied Multi-agent Markov decision process (MMDP) while first identifying the decentralized nature of the problem formulation. With regard to the development of these approximation algorithms, this thesis made the following contributions:

- The basic, Dec-MMDP algorithm was developed which introduced a mechanism for approximating individual teammates using a feature-based state aggregation technique on the full agent model. We analytically showed that this algorithm results in a computational complexity that scaled as $|D|^{n-1}$ in the number of agents n (rather than $|C|^n$ for the case of MMDP, where $D << C$). Also, we empirically showed that the resulting performance was within 10% of that achieved by its centralized counterpart when both were implemented under the PSM scenario.

- An extension to the basic Dec-MMDP formulation was developed where each agent modeled the aggregate set of its teammates rather than each teammate individually. We called this extension the group-aggregate Dec-MMDP (GA-Dec-MMDP). We analytically showed that this extension resulted in a computational complexity that scaled *linearly* in the number of agents n, rather than exponentially. Also, we empirically showed that the resulting performance was within 20% of that achieved by its centralized counterpart when both were implemented under the PSM scenario.

- A method was developed for generating the feature vector, ϕ, used in the approximation algorithms listed above. We showed how the components of the cost/reward function were translated into features, either binary or $(n-1)$-ary. Also, we analytically showed how the resulting problem complexity is affected through construction of these features.

The third and final focus area dealt with implementing and flight-testing the cooperative planning algorithms within iCCA in realistic, multi-agent scenarios. In particular, we investigated the persistent surveillance problem, in which multiple unmanned aerial vehicles (UAVs) and/or unmanned ground vehicles (UGVs) continuously searched a designated region over indefinite periods of time. This problem directly related to a number of applications, including search and rescue, natural disaster relief operations, urban traffc monitoring, etc. These types of missions were shown to be challenging largely due to their extended duration, which increased the likelihood that one or more agents would experience health-related failures over the course of the mission. The planning system was formulated to anticipate and plan for these failures while maximizing mission performance. In order to investigate these issues, this thesis:

- Formulated the persistent surveillance mission as an MMDP, which fully captured the requirement of scheduling agents to periodically move back and forth between the surveillance location and the base location for refueling and maintenance. In addition, we incorporated a number of randomly-occurring failure scenarios (such as sensor failures, actuator degradations and unexpected fuel usage) and constraints (such as the requirement to maintain a communication link between the base and agents in the surveillance area) into the problem formulation. We showed that the optimal policy for the persistent surveillance problem formulation not only properly managed asset scheduling, but also anticipated the adverse effects of failures on the mission and took proactive actions to mitigate their impact on mission performance.

- Formulated the persistent surveillance mission as a Dec-MMDP and as a GA-Dec-MMDP. We empirically showed that the resulting policies properly managed asset scheduling while anticipating the adverse effects of failures and attempted to take proper proactive actions so as to mitigate their impact on mission performance. We further showed that implementing the resulting policies in the persistent surveillance scenario resulted in mission performances within 10% and 20% of the optimal solution, respectively.

- Highlighted the development of an automated battery changing/charging station for hover-capable unmanned aerial vehicles. This technology was shown to be necessary to enable truly autonomous persistent capabilities.

7.1 Future Work

While this thesis primarily considered sources of uncertainty that are independent of system state, an interesting and important extension would be to consider sources of uncertainty that are functions of the state. For example, the rate at which a vehicle burns fuel can be expressed as a function of the aggressiveness of its maneuvers which, in turn, are likely a function of its current task or location. While this extension to incorporate state-dependent parameters adds complexity, it also enables a more general problem statement. In addition, learning state-dependent parameters presents an interesting problem that lends itself well to multi-agent learning methods not covered in this thesis.

Furthermore, the idea of approximating each agentâĂŹs set of teammates while maintaining a full-fidelity model of itself was presented in Chapter 4 as the basis for two new problem formulations. Extending this idea of teammate approximation to the more general Dec-MDP problem formulation is of interest as there is ample literature on the Dec-MDP formulation as well as many published solution approaches for benchmark comparisons.

Finally, one of the major challenges of solving feature-based approximated MDPs, where the features are added as elements of the state, is determining how to generate transition models for these features, since the resulting policy can be extremely sensitive to these models. This topic has received little attention in the literature due to the common use of model-free reinforcement learning methods, which do not explicitly carry a transition model, for solving approximate MDPs. However, when a model is desired, it is particularly challenging to characterize feature state transitions when the features are comprised of a non-intuitive set, or when the features themselves have little intuition to support even a hand-coded heuristic approximation.

Bibliography

[1] M. Alighanbari, L.F. Bertuccelli, and J.P. How. A Robust Approach to the UAV Task Assignment Problem. In *IEEE Conference on Decision and Control (CDC)*, pages 5935–5940, 13–15 Dec. 2006.

[2] M. Alighanbari and J. P. How. Cooperative task assignment of unmanned aerial vehicles in adversarial environments. In *American Control Conference (ACC)*, pages 4661–4666 vol. 7, Portland, OR, 8-10 June 2005.

[3] M. Alighanbari and J. P. How. A robust approach to the UAV task assignment problem. *International Journal of Robust and Nonlinear Control*, 18(2):118–134, January 2008.

[4] M. Alighanbari, Y. Kuwata, and J. P. How. Coordination and control of multiple UAVs with timing constraints and loitering. In *American Control Conference (ACC)*, volume 6, pages 5311–5316 vol.6, 4-6 June 2003.

[5] Mehdi Alighanbari. Task Assignment Algorithms for Teams of UAVs in Dynamic Environments. Master's thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, Cambridge MA, June 2004.

[6] M. Allen and S. Zilberstein. Complexity of decentralized control: Special cases. In *Adv. Neural Inform. Proc. Systems*, volume 22, pages 19–27. Citeseer, 2009.

[7] D. Austin, L. Fletcher, and A. Zelinsky. Mobile Robotics in the Long Term - Exploring the Fourth Dimension. *Procedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2001.

[8] B. Basso, J. Love, and J.K. Hedrick. Airborne, autonomous & collaborative. *Mechanical engineering*, 133(4):26 31, 2011.

[9] R. Becker, S. Zilberstein, and V. Lesser. Decentralized markov decision processes with event-driven interactions. In *Proceedings of the Third International*

Joint Conference on Autonomous Agents and Multiagent Systems-Volume 1, pages 302–309. IEEE Computer Society, 2004.

[10] R. Becker, S. Zilberstein, V. Lesser, and C.V. Goldman. Transition-independent decentralized markov decision processes. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 41–48. ACM, 2003.

[11] J. Bellingham, M. Tillerson, A. Richards, and J. P. How. Multi-task allocation and path planning for cooperating UAVs. In *Cooperative Control: Models, Applications and Algorithms at the Conference on Coordination, Control and Optimization*, pages 1–19, November 2001.

[12] R. Bellman. On the Theory of Dynamic Programming. *Proc. of the National Academy of Sciences of the United States of America*, 38(8):716, 1952.

[13] Richard Bellman. *Dynamic Programming.* Dover Publications, March 2003.

[14] C.A. Bererton, G.J. Gordon, and S. Thrun. Auction mechanism design for multi-robot coordination. In *17th Annual Conf. on Neural Information Processing Systems*, 2003.

[15] Daniel S. Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. The complexity of decentralized control of markov decision processes. *Mathematics of Operation Research*, 27(4):819–840, 2002.

[16] D. Bertsekas. *Dynamic Programming and Optimal Control.* Athena Scientific, Belmont, MA, 2007.

[17] D. Bertsekas and J. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, Belmont, MA, 1996.

[18] Luca F. Bertuccelli. Robust Planning for Heterogeneous UAVs in Uncertain Environments. Master's thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, Cambridge MA, June 2004.

[19] Luca F. Bertuccelli. *Robust Decision-Making with Model Uncertainty in Aerospace Systems.* PhD thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, Cambridge MA, September 2008.

[20] B. Bethke, L. F. Bertuccelli, and J. P. How. Experimental demonstration of adaptive MDP-based planning with model uncertainty. In *AIAA Guidance Navigation and Control*, Honolulu, Hawaii, 2008.

[21] B. Bethke, J. P. How, and J. Vian. Group health management of UAV teams with applications to persistent surveillance. In *American Control Conference (ACC)*, pages 3145–3150, Seattle, WA, 11-13 June 2008.

[22] B. Bethke, J. P. How, and J. Vian. Multi-UAV Persistent Surveillance With Communication Constraints and Health Management. In *AIAA Guidance, Navigation, and Control Conference (GNC)*, August 2009. (AIAA-2009-5654).

[23] Brett M. Bethke. *Kernel-Based Approximate Dynamic Programming Using Bellman Residual Elimination.* PhD thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, Cambridge MA, February 2010.

[24] S. Bhatnagar, R. S. Sutton, M. Ghavamzadeh, and M. Lee. Incremental natural actor-critic algorithms. In John C. Platt, Daphne Koller, Yoram Singer, and Sam T. Roweis, editors, *NIPS*, pages 105–112. MIT Press, 2007.

[25] N. Cattrysse Luk and G. Dirk. A survey of algorithms for the generalized assignment problem. *European Journal of Operational Research*, 60(3):260–272, 1992.

[26] H.-L. Choi, L. Brunet, and J. P. How. Consensus-based decentralized auctions for robust task allocation. *IEEE Transactions on Robotics*, 25(4):912–926, August 2009.

[27] PC Chu and JE Beasley. A Genetic Algorithm for the Generalised Assignment Problem. *Computers & Operations Research*, 24(1):17–23, 1997.

[28] Daniel R. Dale. Automated ground maintenance and health management for autonomous unmanned aerial vehicles. Master's thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, Cambridge MA, June 2007.

[29] P. De, A. Raniwala, R. Krishnan, K. Tatavarthi, J. Modi, N. A. Syed, S. Sharma, and T. Chiueh. MiNT-m: An Autonomous Mobile Wireless Experimentation Platform. *Proceedings of the 4th International Conference on Mobile Systems, Applications and Services*, 2006.

[30] S. de Jong, K. Tuyls, and I. Sprinkhuizen-Kuyper. Nature-Inspired Multi-Agent Coordination in Task Assignment Problems. *Proc. of the 6th European Symposium on Adaptive Learning Agents and MAS (ALAMAS)*, 2006.

[31] E. Saad, J. Vian, G.J. Clark and S. Bieniawski. Vehicle Swarm Rapid Prototyping Testbed. In *AIAA Infotech@Aerospace*, Seattle, WA, 2009.

[32] A. Farinelli, L. Iocchi, and D. Nardi. Multirobot Systems: a Classification Focused on Coordination. Systems, Man and Cybernetics, Part B. *IEEE Transactions*, 34, 2004.

[33] T. Fawcett. *Feature discovery for problem solving systems.* PhD dissertation, University of Massassachusetts, Amherst, 1993.

[34] J. Ferber. *Multi-agent systems: an introduction to distributed artificial intelligence*, volume 222. Addison-Wesley London, 1999.

[35] M.L. Fisher, R. Jaikumar, and L.N. Van Wassenhove. A Multiplier Adjustment Method for the Generalized Assignment Problem. *Management Science*, pages 1095–1103, 1986.

[36] T. Gabel and M. Riedmiller. Reinforcement learning for dec-mdps with changing action sets and partially ordered dependencies. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 3*, pages 1333–1336. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[37] D.W. Gage. Ugv history 101: A brief history of unmanned ground vehicle (ugv) development efforts. Technical report, DTIC Document, 1995.

[38] A. George and W.B. Powell. An adaptive-learning framework for semi-cooperative multi-agent coordination, 2007.

[39] B. Gerkey and M.J. Mataric. Are (explicit) multi-robot coordination and multi-agent coordination really so different. In *Proceedings of the AAAI spring symposium on bridging the multi-agent and multi-robotic research gap*, pages 1–3, 2004.

[40] B.P. Gerkey and M.J. Mataric. A Formal Analysis and Taxonomy of Task Allocation in Multi-Robot Systems. *The Int'l Journal of Robotics Research*, 23(9):939, 2004.

[41] G. Goebel. In the public domain: Unmanned aerial vehicles, 2005.

[42] Claudia V. Goldman and Shlomo Zilberstein. Optimizing information exchange in cooperative multi-agent systems. In *Proc. of the 2nd Int'l Joint Conf. on Autonomous Agents and Multiagent Systems*, pages 137–144, New York, NY, USA, 2003. ACM.

[43] C.V. Goldman and S. Zilberstein. Decentralized control of cooperative systems: Categorization and complexity analysis. *Journal of Artificial Intelligence Research*, 22(1):143–174, 2004.

[44] C. Guestrin, D. Koller, and R. Parr. Multiagent planning with factored mdps. *Advances in neural information processing systems*, 2:1523–1530, 2002.

[45] Y. Hada and S. Yuta. A First-Stage Experiment of Long Term Activity of Autonomous Mobile Robot - Result of Respective Base-Docking Over a Week. *Lecture Notes in Control and Information Sciences: Experimental Robotics VII*, 271:229–238, 2001.

[46] Y. Hada and S. Yuta. A first-stage experiment of long term activity of autonomous mobile robot - result of repetitive base-docking over a week. In *Experimental Robotics VII*, ISER '00, pages 229–238, London, UK, 2001. Springer-Verlag.

[47] E. A. Hansen, D. S. Bernstein, and S. Zilberstein. Dynamic programming for partially observable stochastic games. In Deborah L. McGuinness and George Ferguson, editors, *AAAI*, pages 709–715. AAAI Press / The MIT Press, 2004.

[48] M. J. Hirsch, P. M. Pardalos, R. Murphey, and D. Grundel, editors. *Advances in Cooperative Control and Optimization*, volume 369. Springer, Nov 2007.

[49] J. P. How, B. Bethke, A. Frank, D. Dale, and J. Vian. Real-time indoor autonomous vehicle test environment. *IEEE Control Systems Magazine*, 28(2):51–64, April 2008.

[50] http://itl.nist.gov/div898/handbook/index.htm. Engineering statistics handbook.

[51] http://mikrokopter.us.

[52] http://openjaus.com/understanding-sae-jaus .

[53] http://sparkfun.com/products/3970.

[54] http://www.cdlsystems.com/index.php/stanag4586 .

[55] http://www.jaustoolset.org .

[56] http://www.nellis.af.mil/units/uascenterofexcellence.asp .

[57] N. K. Jong and P. Stone. State abstraction discovery from irrelevant state variables. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 752–757, 2005.

[58] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

[59] K. Kouzoubov and D. Austin. Autonomous recharging for mobile robotics. In *Australian Conference on Robotics and Automation, Auckland*, pages 27–29, 2002.

[60] M. Kroon and S. Whiteson. Automatic feature selection for model-based reinforcement learning in factored mdps. In *Machine Learning and Applications, 2009. ICMLA '09. International Conference on*, pages 324 –330, dec. 2009.

[61] Y. Kuwata and J. P. How. Stable trajectory design for highly constrained environments using receding horizon control. In *American Control Conference (ACC)*, volume 1, pages 902–907 vol.1, Boston, MA, 30 June-2 July 2004.

[62] Y. Kuwata and J. P. How. Robust cooperative decentralized trajectory optimization using receding horizon milp. In *American Control Conference (ACC)*, pages 522–527, 9-13 July 2007.

[63] Y. Kuwata and J. P. How. Cooperative distributed robust trajectory optimization using receding horizon MILP. *IEEE Transactions on Control Systems Technology*, 19(2):423–431, March 2011.

[64] J. Slotine W. Li. *Applied Nonlinear Control.* Prentice Hall, Upper Saddle River, NJ, 1991.

[65] M. L. Littman, T. L. Dean, and L. P. Kaelbling. On the complexity of solving markov decision problems. In *In Proc. of the 11th Int'l Conf. on Uncertainty in Artificial Intelligence*, pages 394-402, 1995.

[66] J. Marschak. Elements for a theory of teams. *Management Science*, pages 127–137, 1955.

[67] S. Martello and P. Toth. Generalized Assignment Problems. *Algorithms and Computation*, pages 351–369, 1992.

[68] H. Brendan McMahan, Geoffrey J. Gordon, and Avrim Blum. Planning in the presence of cost functions controlled by an adversary. In *Proc. of the Twentieth Int'l Conf. on Machine Learning*, 2003.

[69] F. S. Melo and M. Veloso. Decentralized mdps with sparse interactions. *Artificial Intelligence*, 175:1757–1789, 2011.

[70] Francisco S. Melo and Manuela Veloso. Local multiagent coordination on decentralized mdps with sparse interactions. Technical report, Carnagie Mellon University, 2010. http://reports-archive.adm.cs.cmu.edu/anon/anon/2010/CMU-CS-10-133.pdf.

[71] F.S. Melo and M. Veloso. Decentralized mdps with sparse interactions. *Artificial Intelligence*, 2011.

[72] R. Murray. Recent research in cooperative control of multi-vehicle systems. *ASME Journal of Dynamic Systems, Measurement, and Control*, 2007.

[73] R.M. Nauss. Solving the generalized assignment problem: An optimizing and heuristic approach. *INFORMS Journal on Computing*, 15(3):249, 2003.

[74] A. Nedic, A. Ozdaglar, and P.A. Parrilo. Constrained Consensus and Optimization in Multi-Agent Networks. *IEEE Transactions on Automatic Control*, 2009.

[75] T.J. Nugent and J.T. Kare. Laser Power for UAVs, 2008. http://lasermotive.com/wp-content/uploads/2010/04/Wireless-Power-for-UAVs-March2010.pdf.

[76] R. Olfati-Saber, J.A. Fax, and R.M. Murray. Consensus and cooperation in networked multi-agent systems. *Proceedings of the IEEE*, 95(1):215 –233, jan. 2007.

[77] I.H. Osman. Heuristics for the Generalised Assignment Problem: Simulated Annealing and Tabu Search Approaches. *OR Spectrum*, 17(4):211–225, 1995.

[78] L. Panait and S. Luke. Cooperative Multi-Agent Learning: The State of the Art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434, 2005.

[79] C.H. Papadimitriou and J.N. Tsitsiklis. The complexity of Markov decision processes. *Mathematics of operations research*, 12(3):441–450, 1987.

[80] R. Parr, C. Painter-Wakefield, L. Li, and M. L. Littman. Analyzing feature generation for value-function approximation. In *International Conference on Machine Learning (ICML)*, pages 737–744, 2007.

[81] M. Petrik and S. Zilberstein. A successive approximation algorithm for coordination problems, 2010.

[82] S. Ponda, J. Redding, H.-L. Choi, J. P. How, M. A. Vavrina, and J. Vian. Decentralized planning for complex missions with dynamic communication constraints. In *American Control Conference (ACC)*, Baltimore, MD, July 2010.

[83] P. Pudil, J. Novovicová, and J. Kittler. Floating search methods in feature selection. *Pattern Recognition Letters*, 15(11):1119 – 1125, 1994.

[84] M. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* Wiley, 1994.

[85] R. Radner. Team decision problems. *The Annals of Mathematical Statistics*, 33(3):857–881, 1962.

[86] J. Redding, B. Bethke, L. Bertuccelli, and J. How. Active learning in persistent surveillance uav missions. In *AIAA Infotech@Aerospace Conference*, April 2009 (AIAA-2009-1981).

[87] J. D. Redding, T. Toksoz, N. Kemal Ure, A. Geramifard, J. P. How, M. Vavrina, and J. Vian. Persistent distributed multi-agent missions with automated battery management. In *AIAA Guidance, Navigation, and Control Conference (GNC)*, August 2011. (AIAA-2011-6480).

[88] Liran Katzir Reuven Cohen and Danny Raz. An efficient approximation for the generalized assignment problem. *Information Processing Letters*, 100(4):162–166, 2006.

[89] A. Richards, J. Bellingham, M. Tillerson, and J. P. How. Coordination and control of multiple UAVs. In *AIAA Guidance, Navigation, and Control Conference (GNC)*, Monterey, CA, August 2002. AIAA Paper 2002-4588.

[90] G.T. Ross and R.M. Soland. A Branch and Bound Algorithm for the Generalized Assignment Problem. *Mathematical programming*, 8(1):91–103, 1975.

[91] S. Russell and P. Norvig. *Artificial Intelligence, A Modern Approach.* Prentice Hall, 2003.

[92] M. Savelsbergh. A Branch-and-Price Algorithm for the Generalized Assignment Problem. *Operations Research*, 45(6):831–841, 1997.

[93] Ketan Savla, Tom Temple, and Emilio Frazzoli. Human-in-the-loop vehicle routing policies for dynamic environments. In *IEEE Conf. on Decision and Control*, 2008.

[94] E. Semsar-Kazerooni and K. Khorasani. Multi-agent team cooperation: A game theory approach. *Automatica*, 45(10):2205–2213, 2009.

[95] Sven Seuken and Shlomo Zilberstein. Formal models and algorithms for decentralized decision making under uncertainty. *Autonomous Agents and Multi-Agent Systems*, 17(2):190–250, 2008.

[96] J. Shen, V. Lesser, and N. Carver. Minimizing communication cost in a distributed bayesian network using a decentralized mdp. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 678–685. ACM, 2003.

[97] D.B. Shmoys and É. Tardos. An Approximation Algorithm for the Generalized Assignment Problem. *Mathematical Programming*, 62(1):461–474, 1993.

[98] M. C. Silverman, B. Jung, D. Nies, and G. S. Sukhatme. Staying alive longer: Autonomous robot recharging put to the test. Technical Report CRES-03-015, Center for Robotics and Embedded Systems (CRES), University of Southern California, 2003, 2003.

[99] M. T. J. Spaan, G. J. Gordon, and N. A. Vlassis. Decentralized planning under uncertainty for teams of communicating agents. In Hideyuki Nakashima, Michael P. Wellman, Gerhard Weiss, and Peter Stone, editors, *AAMAS*, pages 249–256. ACM, 2006.

[100] M.T.J. Spaan and F.S. Melo. Interaction-driven markov games for decentralized multiagent planning under uncertainty. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 1*, pages

525–532. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[101] P. Stone and M. Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383, 2000.

[102] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, 1998.

[103] R.S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[104] K.A. Swieringa, C.B. Hanson, J.R. Richardson, J.D. White, Z. Hasan, E. Qian, and A. Girard. Autonomous battery swapping system for small-scale helicopters. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3335–3340, May 2010.

[105] M. Valenti, B. Bethke, G. Fiore, J. P. How, and E. Feron. Indoor Multi-Vehicle Flight Testbed for Fault Detection, Isolation, and Recovery. In *AIAA Guidance, Navigation, and Control Conference (GNC)*, Keystone, CO, August 2006 (AIAA-2006-6200).

[106] M. Valenti, B. Bethke, J. P. How, D. P. de Farias, and J. Vian. Embedding Health Management into Mission Tasking for UAV Teams. In *American Control Conference (ACC)*, pages 5777–5783, New York City, NY, 9-13 July 2007.

[107] M. Valenti, D. Dale, J. How, and J. Vian. Mission health management for 24/7 persistent surveillance operations. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, Myrtle Beach, SC, August 2007.

[108] Mario J. Valenti. *Approximate Dynamic Programming with Applications in Multi-Agent Systems.* PhD thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, Cambridge MA, May 2007.

[109] V. Vladimerouy, A. Stubbs, J. Rubel, A. Fulford, J. Strick, and G. Dullerud. A hovercraft testbed for decentralized and cooperative control. In *American Control Conference (ACC)*, pages 5332–5337, Boston, MA, July 2004.

[110] L. G. Weiss. Autonomous robots in the fog of war. *IEEE Spectrum*, 48(8 (NA)):30, August, 2011.

[111] S. Whiteson, P. Stone, K. O. Stanley, R. Miikkulainen, and N. Kohl. Automatic feature selection in neuroevolution. In *Genetic and Evolutionary Computation Conference*, pages 1225–1232, 2005.

[112] J. Wu and E.H. Durfee. Mixed-integer linear programming for transition-independent decentralized mdps. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 1058–1060. ACM, 2006.

[113] P. Xuan, V. Lesser, and S. Zilberstein. Communication decisions in multi-agent cooperation: Model and experiments. In *Proceedings of the fifth international conference on Autonomous agents*, pages 616–623. ACM, 2001.

[114] P. Xuan, V. Lesser, and S. Zilberstein. Modeling cooperative multiagent problem solving as decentralized decision processes. *Autonomous Agents and Multi-Agent Systems*, 2004.

[115] S. Zilberstein, R. Washington, D. S. Bernstein, and A.-I. Mouaddib. Decision-theoretic control of planetary rovers. In Michael Beetz, Joachim Hertzberg, Malik Ghallab, and Martha E. Pollack, editors, *Advances in Plan-Based Control of Robotic Agents*, volume 2466 of *Lecture Notes in Computer Science*, pages 270–289. Springer, 2001.